A STUDY OF THE DESCRIPTION MATCHING PROBLEM

by

David Alan Spencer

B.S.E. Princeton University

(1969)

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1971

Signature of Author________________________________
Department of Electrical Engineering, May 14, 1971

Certified By________________________________
Thesis Supervisor

Accepted By________________________________
Chairman, Departmental Committee on Graduate Students

A STUDY OF THE DESCRIPTION MATCHING PROBLEM

by

David Alan Spencer

Submitted to the Department of Electrical Engineering
on May 14, 1971, in partial fulfillment of
the requirements for the Degree of Master of Science.

ABSTRACT

Descriptive mechanisms have been an important component of much recent work in artificial intelligence. This thesis investigates several aspects of descriptions through discussion of a particular type of description which is simple enough to be easily discussed and yet appears able to support fairly complex behavior. The idea of description comparison, for the purpose of discovering the differences between two descriptions, is emphasized. Several general methods are presented for performing this comparison operation on the particular type of description discussed in this thesis, and their merits are discussed relative to each other and relative to methods which make more use of knowledge about the application in which the descriptive methods are to be used.

THESIS SUPERVISOR: Patrick H. Winston
TITLE: Assistant Professor of Electrical Engineering

ACKNOWLEDGEMENT

I would like to thank Patrick Winston for the idea for this study and for supervising this thesis, Professor Joseph Weizenbaum for valuable comment and criticism, Jeffrey Kaplan for encouragement and useful discussions, and David Pensak for providing the chemistry examples and a short introduction to organic chemistry.

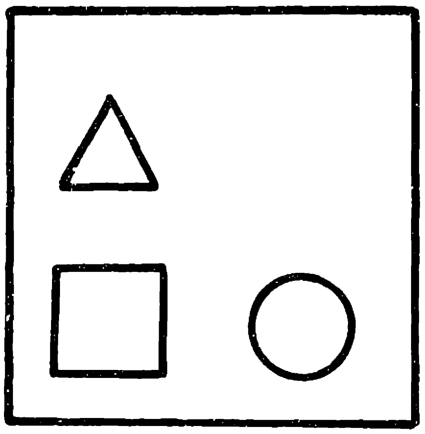
Work reported herein was supported by the Artificial Intelligence Laboratory, an M.I.T. research program sponsored by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-70-A-0362-0002.

TABLE OF CONTENTS

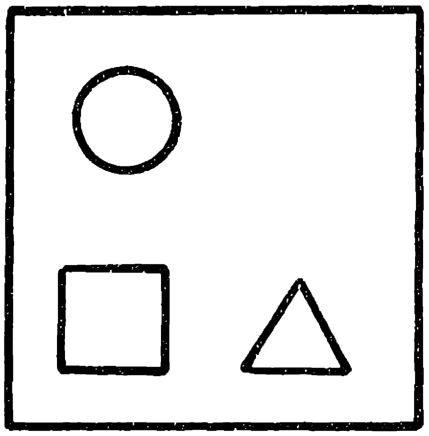
1. Introduction

Many programs produced in the course of artificial intelligence research have involved the use of internal descriptions of some problem area, often descriptions of scenes or diagrams. An operation often performed with these descriptions is description comparison, the purpose of which is to discover the differences between two descriptions. In many cases these descriptions and especially this comparison operation are not recognized as being such, however, or their importance is not emphasized. As a result the details are not discussed or they are given as a collection of procedures especially designed for the application and seemingly having little content of more general value. This research was undertaken in the hope that something significant of a more general nature could be said about descriptions, and particularly about description comparison. This thesis investigates several aspects of descriptive mechanisms through the discussion of a particular form of description. The description format chosen for study has the advantage that a fairly straightforward form of description comparison, involving an operation called description matching, can be formulated for it.

Something of a foretaste of the difficulties involved in description comparison can be gained from trying to compare the two figures below. What are the differences between these



(a)



(b)

two figures? In (a) the triangle is above the square, while in (b) it has been moved to the right of the square, changing places with the circle. But look again. Is it not equally true to say that the object above the square has been changed from a triangle to a circle, and that likewise the circle to the right of the square in (a) is changed to a triangle in (b)? These are two distinctly different descriptions of the differences between these figures. The first is based on the idea that the triangles in both scenes are "the same" and likewise for the circles, and the second is based on the idea that what is really important is the objects' relations to the square, the squares in the two figures being identified. The operation being performed is one of pairing or matching corresponding objects in the two figures. Which pairs of objects correspond to one another depends on the properties of these objects, on their relationships to one another, and on some measure of of the relative importance of these properties and relationships.

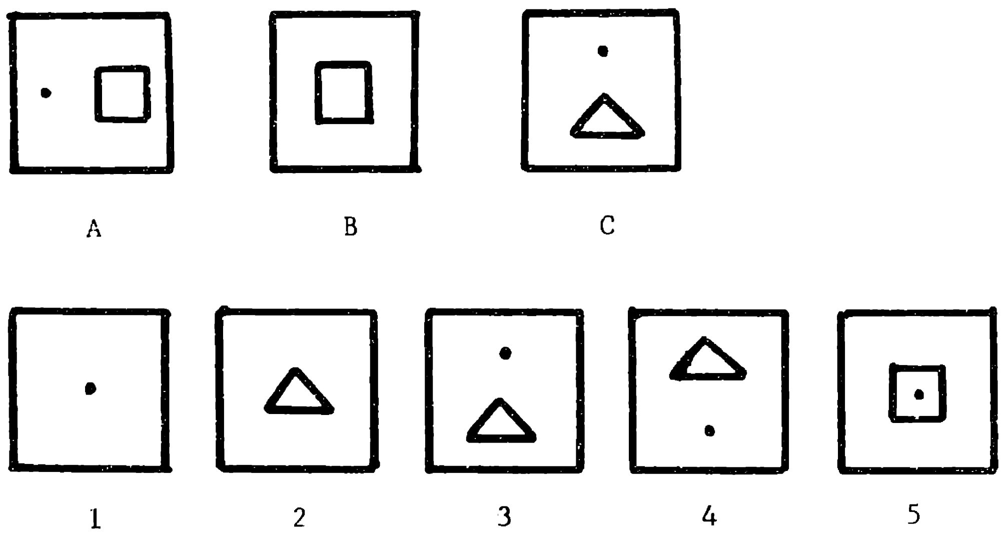
In the next chapter two programs which make extensive use of descriptions and description comparison are discussed, in order to give some idea of how description mechanisms are used, and also in order to provide examples for later discussion. In the third chapter the specific description format to be discussed is presented and a formulation of a type of description comparison for this format is given. This is followed by a chapter which discusses some methods for performing this comparison operation. These methods make use of knowledge about the structure of this description format, but try not to depend on knowledge about any particular application area. In the fifth chapter there is a short discussion of some experiments performed with an actual implementation of some of these ideas, followed by some conclusions. It is decided that while some things can be said in general about descriptions, and some context free mechanisms can be formulated, a knowledge of the context of application seems to be essential to the design and use of a successful description mechanism.

II. Previous Uses of Description Comparison

There are a great many programs written for the purposes of artificial intelligence research which could be said to use internal descriptions of one sort or another, and several of these could perhaps be said to use some sort of description comparison. However, two programs stand out because of their explicit use of somewhat more general descriptive mechanisms and because of the complexity of the description comparison they require. These are Evans' analogy problem solver [1] (numbers in brackets refer to the bibliography) and Winston's program for learning structural descriptions of concepts [6]. In order to get some idea of the descriptive mechanisms used in these two programs a summary of both of them is given below.

Evans' Analogy Problem Solver

The purpose of Evans' program is to solve analogy problems such as are used on college entrance examinations. An example is given below. The problem is to find which of the five answer figures is related to figure C in the same way as figure B is to figure A. This is to be done, supposedly, by finding a rule which will take figure A into figure B and applying this rule to figure C to see which answer results.



In his program which solves such problems, Evans makes use of several descriptive formats. Initially the various figures are given to the program described in terms of line segments, the endpoints and curvatures of these segments being given. Using this initial description the program constructs a new description which breaks each figure up into its several components. This is done mainly on the basis of connected groups of lines, but the program is also able to break up connected figures into sub-components when desirable. Then the relationships among the various objects within a figure are computed. Such relations might include the fact that one object is above another, or that one is inside another, and so on. The final step of the description building process is to find similarities between objects in the various figures. Two objects are similar if they can be transformed into one another by appropriate rotations, reflections, and changes in scale. All such similarities that might be required for the solution process are computed, both between objects within a figure and between objects in different figures.

Evans describes the next stage of the solution process as finding one or more rules which transform figure A into figure B. Each of these rules is then generalized to the point where it will also transform figure C into one of the answer figures. A "best" rule is chosen and the figure into which it takes C is chosen as the answer.

The process by which this is done in fact involves several stages of description comparison. The first descriptions to be compared are those of figures A and B, these being the descriptions in terms of objects, relations among objects, and similarity transformations built up during the previous processing. The comparison consists of matching all objects in figure A with those objects in figure B which are similar to them. If there are several alternative matchings, as there often are, all such possible matchings are found. Each matching defines a rule for transforming figure A into figure B, this rule consisting of a statement of the transformations required to take all matched objects in A into their matching objects in B. If an object in A is not matched, it is considered to be deleted by the transformation to figure B. Similarly, unmatched objects in B are said to have been added to the figure by the transformation.

The same description comparison operation, involving the matching of similar objects, is performed between the descriptions of figure C and each of the answer figures, and a description of the required C to answer transformation is generated. A second order description comparison is made, which involves matching the objects in the A to B transformation rules with corresponding objects in all transformations from C to an answer figure. This means that objects transformed by some similarity relation in the A to B transformation must be matched in turn with objects that also require some similarity transformation in any C to answer transformation. Added objects must be matched with added objects, and removed objects must be matched with removed objects. This correspondence between the objects of the A to B transformation with objects in the C to answer figure transformation defines a new transformation rule from C to the answer figure. This new transformation is the A to B transformation applied to the corresponding objects in C and the answer figure. However, included in the transformation are the relations among the various objects, such as that the removed object is inside the one to be rotated, and the relations which hold in the two sample figures A and B may not hold in figure C or the answer figure. Therefore, in order to be a correct transformation, the rule must be weakened by removing all relational statements which do not hold true in figure C or the answer figure. The final result is a transformation which takes A to B and C to an answer figure. All such transformations are found, and the "best" is chosen, "best" corresponding to a measure of rule strength or how specific a rule is in stating the transformation. The answer figure chosen is the one into which C is transformed under this transformation.

Winston's Concept Learning Program

Winston also uses descriptions expressed in terms of relationships between objects, although in his case "objects" can be abstractions such as "cube" or "arch," or relationships themselves, such as "above." His idea of description comparison is also similar to Evans', in that it involves matching objects in one description with objects in the other. The basis on which this matching is performed is more sophisticated in some respects than that used by Evans, however. Winston's matching process will be discussed in some detail in a later chapter.

The importance of this description comparison operation is explicitly recognized by Winston as being central to the operation of his procedures. This is in distinction to Evans, who does not appear to view his description comparisons as being such. In fact, Winston reformulates the analogy problem solving process as one of comparing descriptions and finding differences and uses this formulation and his description comparison procedures to solve analogy problems.

THe learning process is the major part of Winston's thesis, however. In basic outline it is quite simple. First the concept learning program is presented with a description of an object which is an instance of the concept to be learned. An example of the sort of concept learned by this program would be a "tower," meaning a stack of cubes. Next the program is presented with descriptions of objects which miss being instances of the concept by only a few features - - certain crucial components are missing, or certain necessary relationships do not hold, or features are added which prevent the description from being an instance of the concept. These descriptions are compared against the description of the concept as it has been learned to that point (initially the description of the first example is used). If the new descriptions are sufficiently like the concept, the program will recognize them as instances of the concept. When told that they are not instances, it looks at the differences found between the concept description and the object description and proposes one or more "theories" as to which differences are likely to have caused the rejection. The appropriate relations in the concept are marked as being critical. This is done by changing them to MUST-BE or MUST-NOT-BE emphatic relations. For example, ABOVE might be changed to MUST-BE-ABOVE. When further scenes are presented to the recognizer, they are rejected if any of these emphatic satellite relations, as they are called, are not found to hold between corresponding objects.

Thus, over several examples, the concept learner gradually discovers which relations must or must not hold in order for an object to be an instance of the concept being learned. It also learns that other relationships are not essential and so leaves them in non-emphatic form or may discard them. In this way the class of object descriptions which the recognizer will accept as fitting the concept description is gradually adjusted to correspond more closely to the ideas of its teacher.

Description comparison is thus used to find the differences between descriptions. In order to do this, however, it must match objects in the two descriptions in such a way that the two descriptions are made to correspond as closely as possible. If some criterion like this were not used, any matching of objects would be reasonable, and a great many more differences would be found than actually exist.

Another way of looking at this is that the number of differences found should be minimized.

Viewing the matching process in this way, it is easy to see how it could be used to locate an instance of a concept in a large description. By performing matching in such a way that the differences found are minimized, the objects in the concept will be matched with objects in the description in such a way that as many relationships as possible hold in both the concept and the matched part of the description. If all the critical (emphatic) relations are found to hold in the description then an instance of the concept has been found. Winston demonstrates such a locating ability in his thesis.

A similar locating ability using a form of description comparison appears in Winograd's program to understand natural language communication [5]. There the problem is that when "the green block to the left of the yellow pyramid" is mentioned, it is desirable to look in the program's knowledge of its world to find out just what block is being referred to, if indeed any such block exists. In order to do this a description is built up in the form of a program in the Micro-Planner implementation of the Planner language [2, 4], and this program/description is executed in order to find the desired block, if it exists. The data base searching operation which this entails is like locating the desired block description within a large description of the program's knowledge of the world.

III. Descriptions and Description Comparison

In the previous chapter several programs were discussed which make use of descriptive techniques. In this chapter descriptions will be dicussed with a view toward formalizing an idea of description comparison. It is concluded that this is difficult, if not impossible, for the general class of programming constructs that could be considered descriptions, and so a particular type of description is defined and a notion of description comparison formalized for it.

Uses of Descriptions

First, then, what uses have been made of constructs which might be included under an (intuitive) idea of descriptions? The previous chapter gives several specific examples, and the usage there seems to fall into two categories:

a)Storage of information

b)Description comparison

Furthermore, within the latter category various subdivisions can be recognized, such as recognition processes, learning processes, locating objects, and analogy problem solving. There are undoubtedly other usages that could be added to this list. The idea of description comparison seems fundamental to the usage of descriptive techniques, and in this thesis we are more interested in the comparison aspect of descriptions than in the information storage aspect. Therefore the uses of description comparison mentioned above will be discussed in more detail.

A recognition process begins with a basic description in terms of some primative descriptive elements. For example, in scene recognition, this basic description might consist of a matrix of 1's and 0's, 1's representing black picture elements and 0's representing white ones. Another form of initial description for such a process might consist of a scene represented by lines, their endpoints and curvatures. The result of a recognition process could consist of a statement like "This is a ---" or of a complex, structured description of the objects found and their relationships. The intermediate stages involve what could be called description building. The recognition program has available to it descriptions of the various concepts with which it is to deal, and certain of these are compared with the initial description. Depending on the results of these comparisons - what differences were found - new information may be added to the description (or, alternatively, a new description constructed) and a choice made of what concept descriptions to compare with the resulting description. This continues until a description of the type desired as output is achieved.

A concrete example of such a process is proposed by Winston in his thesis. A vidisector camera is used to observe a scene. Its input is used by programs which find lines (edges of polyhedral objects) in the scene, producing a description in terms of these lines. This description is in turn used as input to a set of programs which give as output a scene description in terms of regions within the field of view (corresponding to the faces of the objects) which are grouped into objects. The relationships among these objects are then found, groupings of objects recognized, and finally perhaps the entire scene is recognized as an instance of some concept description, such as a house, a tower of blocks, or whatever. Notice that at each stage there is information loss as irrelevant details are absorbed into more abstract descriptions. The value of this is that programs which then use these scene descriptions (such as programs to operate a mechanical arm) can deal with these simpler, more abstract descriptions. Of course, information loss is not essential, in that the original description can be kept so that if information is needed that does not appear in the higher level descriptions it can be computed from the original.

Learning processes are closely related to recognition processes, in that the concept descriptions mentioned previously are what is learned. Roughly, a learning process consists of performing a recognition process, followed by a comparison of the results with the "correct" answers or at least with an indication of whether the results are correct or not, followed in turn by a modification of the recognition process. This modification might consist of changes to the concept descriptions, as in Winston's thesis, or changes to the control information of the process, such as which concept descriptions to try next or what new descriptive information to add under various circumstances. These modifications must be based on the differences between the result and the correct answer, if that is available, or on those differences found but ignored when the concepts were compared with the given descriptions. Thus, what is learned and how depends on just what differences are found during description comparison.

Location of objects within a description is the same sort of operation as comparing a concept description with an initial description in recognition processes, and need not be discussed further. Analogy problems, however, are not so directly related to recognition, and if anything depend more on the idea of differences between descriptions. Winston formulates the problem of solving problems like those used by Evans as one of first describing the differences between the various pairs of figures and then picking the answer whose difference description is most similar to the difference description produced when the first two sample figures are compared. With this formulation of the problem he then uses a description comparison operation (very similar to that to be defined later) to actually solve several analogy problems.

Descriptions

What then constitutes a description for programming purposes? The answer, unfortunately for our needs, is that any data structure can be considered a description and any program can be considered either a description or to operate on descriptions. When a programmer designs a program he devises some sort of scheme for representing (describing) the things the program is to work with. A matrix, for example, might be used as a description of a directed graph (a 1 in position i,j indicating a path from node i to node j), or, as mentioned above, this matrix might be used to represent a scene. Additional description goes into the programs written to use the data. For example, a program might be written to use the matrix description of a directed graph in order to determine if there is a path between two given nodes. This program in some sense constitutes a description of the concept of a path between nodes.

Much more elaborate descriptions are used in the programs mentioned in the previous chapter, but again descriptive information appears in many forms. Evans uses descriptions of figures in terms of lines, endpoints, and curvatures, and these descriptions are in turn represented by particular list structures. However, he also has a subprogram which determines whether two figures are similar in the sense that they can be transformed into each other by rotations, translations and changes of scale. This program thus constitutes a description of the concept of similarity. Similar situations arise in all the other programs mentioned.

This variety of description representations makes it difficult, probably impossible, to talk formally about description comparison in general. Any discussion of this sort would eventually lead to considerations of how to compare two programs representing two concepts (of similarity of figures, say) in order to discover the differences between the two concepts. A solution to this would enable the determination of the equivalence of two programs, a problem known to be unsolvable in general, and certainly difficult in almost all cases where it can be done.

A Particular Description Representation

Since, by the above arguments, description comparison must be discussed in terms of particular description representations, we will define a type of description similar to some of the description representations used by the programs mentioned in the previous chapter. This description representation is limited in the range of things that can be described, but appears able to support non-trivial behavior. A major advantage of this type of description is that a clear formulation of description comparison, called description matching, can be devised for it.

A description, from now on, will be taken to consist of:

a) A set of relational statements of the form

(rel ob1 ob2 ... obn)

where rel is the name of some relationship that holds among the n objects. The term elements will be used to denote both relations and objects.

b) A list of variables for the description, which designates certain of the elements appearing in the description as variables.

c) An evaluation function which maps from the power set of the set of relational statements to some real interval, say (0, 1).

Parts (b) and (c) will be seen to be important only for concept descriptions.

An example of a relational statement is (ABOVE A B). The order of the objects is important, (ABOVE B A) having an entirely different meaning. The use of the word "object" is not intended to signify that these necessarily represent physical objects. Relation names can also be used as objects, as in the relational statement (OPPOSITES RIGHT-OF LEFT-OF), where (RIGHT-OF A B) would be a relational statement indicating that A is to the right of B in the scene. A property might also be used as an object, for example in (IS JOHN GREEDY), although it might be preferable to represent the latter statement as (GREEDY JOHN). Objects have no internal structure, being merely names. Any descriptive structure relevant to an object is explicitly represented as relational statements.

A matching of two descriptions is defined by a one to one mapping from the elements of one description to the elements of the other. It will be found desirable to define one description as the "dominent" one, and the mapping is defined as taking elements of this description into elements of the "inferior" description. Allowable mappings are restricted by the variable list. Elements on the variable list can be mapped to any element of the other description, but all other elements must be mapped to themselves (they are constants). The mapping can be partial in that it need not associate every element of either description with an element of the other. Two elements associated in this way will be said to be paired.

If two relational statements, one in each description, are such that their corresponding elements are paired (this means they must both have the same number of elements) then these relational statements (relations for short) are said to be matched. This means that the same relation holds among corresponding objects in both descriptions. For any given matching, some subset of the relations of a description will be matched. This subset is used as an argument for the evaluation function which returns a value that is taken to be an indication of the quality of the match, 1 being best. The object of description matching, then, is to find a "best" match, i.e., that match which gives the largest evaluation score. Intuitively, this best match is intended to pair those elements of the two descriptions which "most nearly" correspond to each other, in the sense that they are related to the other elements in similar ways.

Notice that the evaluation function adds to the asymmetry of this comparison operation, in that two descriptions being matched will have two evaluation functions, and the matching which maximizes one will not necessarily maximize the other. Thus a decision must be made as to which evaluation function to use, and the description whose function is used is the dominent description. Another source of asymmetry is that constants of the dominent description must be paired with themselves, while constants of the inferior description may be paired with variables of the dominent description.

An example would probably be helpful at this point. The concept description below is one possible representation of the concept of connected lines. The idea represented is that

```
(IS  A  LINE)
(IS  B  LINE)
(HAS-ENDPOINT  A  AEND)
(HAS-ENDPOINT  B  BEND)
(X-COORD  AEND  X)
(X-COORD  BEND  X)
(Y-COORD  AEND  Y)
(Y-COORD  BEND  Y)

Variables = (A, B, BEND, AEND, X, Y)
```

two lines are connected if they have endpoints with identical coordinates. The evaluation function could be a Boolean AND, since all the relations must be matched in order that two lines be connected. A description which would fit this concept is given below. The pairings would be A-Z1, B-Z2,

```
(IS  Z1  LINE)
(IS  Z2  LINE)
(HAS-ENDPOINT  Z1  ZE1)
(HAS-ENDPOINT  Z2  ZE2)
(X-COORD  ZE1  0.5)
(X-COORD  ZE2  0.5)
(Y-COORD  ZE1  1.0)
(Y-COORD  ZE2  1.0)
```

AEND-ZE1, BEND-ZE2, X-0.5, Y-1.0, and since all other elements are constants they would be paired with themselves. After this matching a statement of the form (CONNECTED Z1 Z2) might be added to the description. This would be part of a recognition process, however, and does not fall within the domain of the descriptions and matching defined above.

To summarize, then, the function of the relational statements is to list the relationships among a set of objects. In the case of concept descriptions some of these objects, and some of the relation names, may be represented by variables. The evaluation function incorporates information about which are the important relations in a concept and about the interactions among these relations, and provides a means of measuring the quality of a match between two descriptions. A "best" match is intended to give pairings between those objects which are most nearly similar in the two descriptions, similarity being determined by matched relations. Since many matches are possible between two descriptions, the evaluation function is used to determine which relations are the most important to match, and best match is defined in terms of maximizing the value of the evaluation function.

Finally, a note on implementation. It is not likely that a scheme of this sort would be best implemented as simply an unstructured list of relations with associated variable list and evaluation function. Depending on the application in which this descriptive technique was being used, relations might be represented in many ways. In the case mentioned above where directed graphs were being described, the matrix representation might well be the most efficient, even though the same information might be represented by statements of the form (PATH I J) to indicate a path from node I to node J. The matrix represents the same information, but in a more efficient form for any likely applications. Description matching can still be thought of in the same terms, using the correspondence between a 1 in entry i,j of the matrix and the relational statement above. Certain other operations become more efficient, however. For example, matrix multiplication can be used to find if any path exists between two given nodes.

Difficulties With This Type of Description

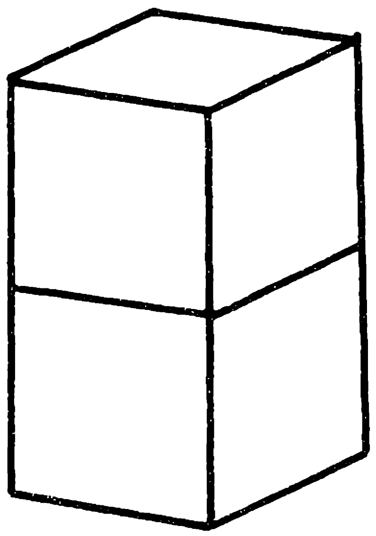
It was stated previously that this description form is not adequate for all descriptive purposes. This is probably obvious, but some of the problems will be mentioned in this section anyway.

One difficulty has to do with the finiteness of these descriptions. For example, it might be desirable to have a concept of a picket fence in some scene recognizer. A fence can have an arbitrary number of pieces, each identical and each related to its neighbors in a very specific way. This concept of a picket fence could not be represented adequately within this descriptive technique, however, as it requires an arbitrary number of objects, one for each picket, at least. One way to get around this is mentioned by Winston. Rather than have a variable for each picket, he would use one variable to represent a typical picket, and show its relations with one or two other typical pickets. These typical elements would then be structured into a "group" which could represent an arbitrary number of such typical elements. However, in order to match this concept with a scene description, the scene would have to be described in the same terms. This therefore merely pushes the problem away one level, as the description mechanism defined above could not be used to recognize (with one matching) when a group description would be appropriate. However, a recognition process could be defined by constructing a concept description for a typical fence element and repeatedly matching this against the initial scene description, adding new relations to the initial description at each match to show what new objects have been included in the fence being recognized. The additional control mechanism and operations required by the recognition process (instructions as to which concept description to try next, the operation of adding relations to the description) do not fall within our definition of description matching, however. In this sense, description matching as it was defined above is like the pattern matching part of a string processing language. It does not include anything that corresponds to the replacement or goto parts of a statement in such a language.

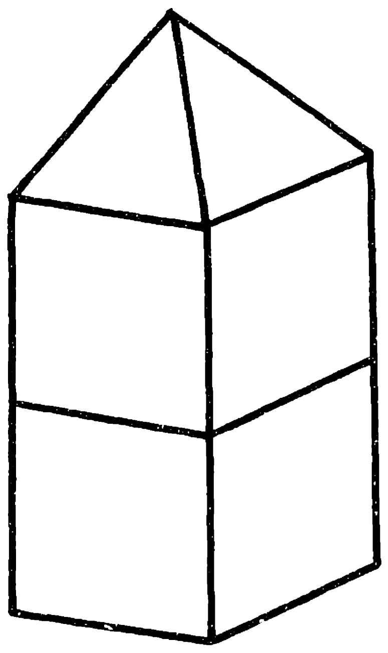
"Connected figure" in a line drawing is another example of a concept that cannot be expressed, for much the same reasons. In general, then, this mechanism cannot represent concepts requiring an arbitrary number of variables. Concepts requiring sequential recognition processes cannot be represented. Also, concepts requiring operations or computations on objects cannot be represented. For example, it might be desirable to have a relational statement like (EQUAL (SUM A B) C), where the sum of A and B would be formed and then the EQUAL comparison made. This is not representable within this description mechanism.

A somewhat more surprizing example of a concept not representable within this description format can be given. Suppose it were desired to define the concept of a "squared tower" to consist of two cubes, one on top of the other, their edges aligned, and with no pyramid resting on top of them. An example is shown in Figure 1 (a) , and 1(b) shows a tower that is not squared. 1(c) shows a first attempt at a description of this concept. The evaluation function for this description would be such that if the two relations involving C were matched, a score of 0 would result.

This does not give the desired result, however. Remember that the object of matching is to maximize the evaluation



(a) Squared Tower



(b) Not a Squared Tower

(IS A CUBE)

(IS B CUBE)

(SUPPORTED-BY B A)

(ALIGNED B A)

(IS C PYRAMID)

(SUPPORTED-BY C B)

Variables = (A, B, C)

(c) Attempt at Concept Description

Figure 1

function. This certainly will not be done if variable C is paired in such a way that the two relations in which C is used are matched. Therefore C simply will not be paired. Figure 1(b) will be accepted as a squared tower since it contains a subfigure which is a squared tower, and the pyramid will be ignored.

Can this difficulty be gotten around somehow? I think not. The root of the problem is that it is desirable that the two relations involving the variable C be matched if at all possible, but that the match fail if they are matched. It would seem desirable to have two evaluations, then. The first would increase the score if the two relations involving C were matched, and this would be the evaluation function used during matching, assuring that these relations would be matched if possible. The second function would then give a zero score if they were matched, indicating that this was not a squared tower.

This does not give the desired behavior when trying to locate a squared tower in a scene consisting of a squared tower and a tower with a pyramid, however. The match program will find the highest evaluation score when pairing the variables in the concept description with the objects in the description of the tower with a pyramid, and only when the second evaluation function is applied will it be found that a mistake was made. Some sort of backup could then be performed, but both this backup and the second evaluation function are outside the scope of the description mechanism being considered here.

The Matching Problem

In this chapter a form of description representation and an idea of description comparison based on that representation have been presented. Some of the failings of this description mechanism have been mentioned. The advantage of this simple form of description is that it permits us to get a handle on the idea of description comparison. The actual process by which the "best" match is found will be the subject of the next chapter.

IV. Some Methods

In the previous chapter the problem of description matching as it is being considered in this thesis was formalized. In this chapter some methods for performing the matching operation are presented.

Guaranteed Best Match

The problem of matching two descriptions is finite as described, in the sense that each description contains a finite number of elements and each element in one description can be paired with elements from the other description in only a finite number of ways. Therefore it is possible to guarantee that a best match will be found by systematically looking through all possible matchings, calculating the evaluation function value for each, and picking the match which gives the highest score. This obviously can involve a large amount of computation, however.

A matching is defined by a map from one set of elements to the other. Assume for simplicity that all elements of one description can be paired with all elements of the other. Let S equal the number of elements in the smaller of the two descriptions, and let L be the number in the larger description. Then in order to try all mappings, including all partial mappings, every subset of the smaller set of elements must be paired with all equal sized subsets from the larger description. For any subset of size i there will be

$$\binom{S}{i}$$

ways to pick a subset of that size from the smaller set. Order this subset in any way. Then by picking all subsets of the same size from the larger set and permuting their members, all possible matches of i elements will be obtained. Since there are

$$\frac{L!}{(L - i)!}$$

ways of picking permutations of i objects from L objects, this gives

$$\frac{S!\ L!}{(S - i)!\ i!\ (L - i)!}$$

possible matches of subsets of size i. Summing over i, there are

$$\sum_{i=2}^{S} \frac{S!\ L!}{(S - i)!\ i!\ (L - i)!}$$

possible matches that would have to be tried using this method. The sum is started at $I=2$ because $I=0$ and 1 give no matched relations, and so are meaningless. For example, if $S=L=5$, not very large sizes at all, the total number of matches from the above formula is 1520. A graph of this function for several typical values of L, and with S fixed at 5, is given in figure 1.

This amount of effort is totally unnecessary, however. By using the information provided by the relational structure of the descriptions, the amount of work required for a guaranteed best match can be substantially reduced. It only pays to pair two elements if that pairing can result in a relation being matched, as the evaluation function depends only on the matched relationships. By using this observation, a great many useless pairings can be eliminated. This leads to a matching method based on pairing relations rather than the pairing of objects. This method will first be described, and then some examples will be given. The description whose evaluation function is being used for the match will be called the dominent description, the other being the inferior description.

The first operation is to form a list containing all possible pairs of relations which might be matched. Recall that a relation from the dominent description will match one from the inferior description only if they both contain the

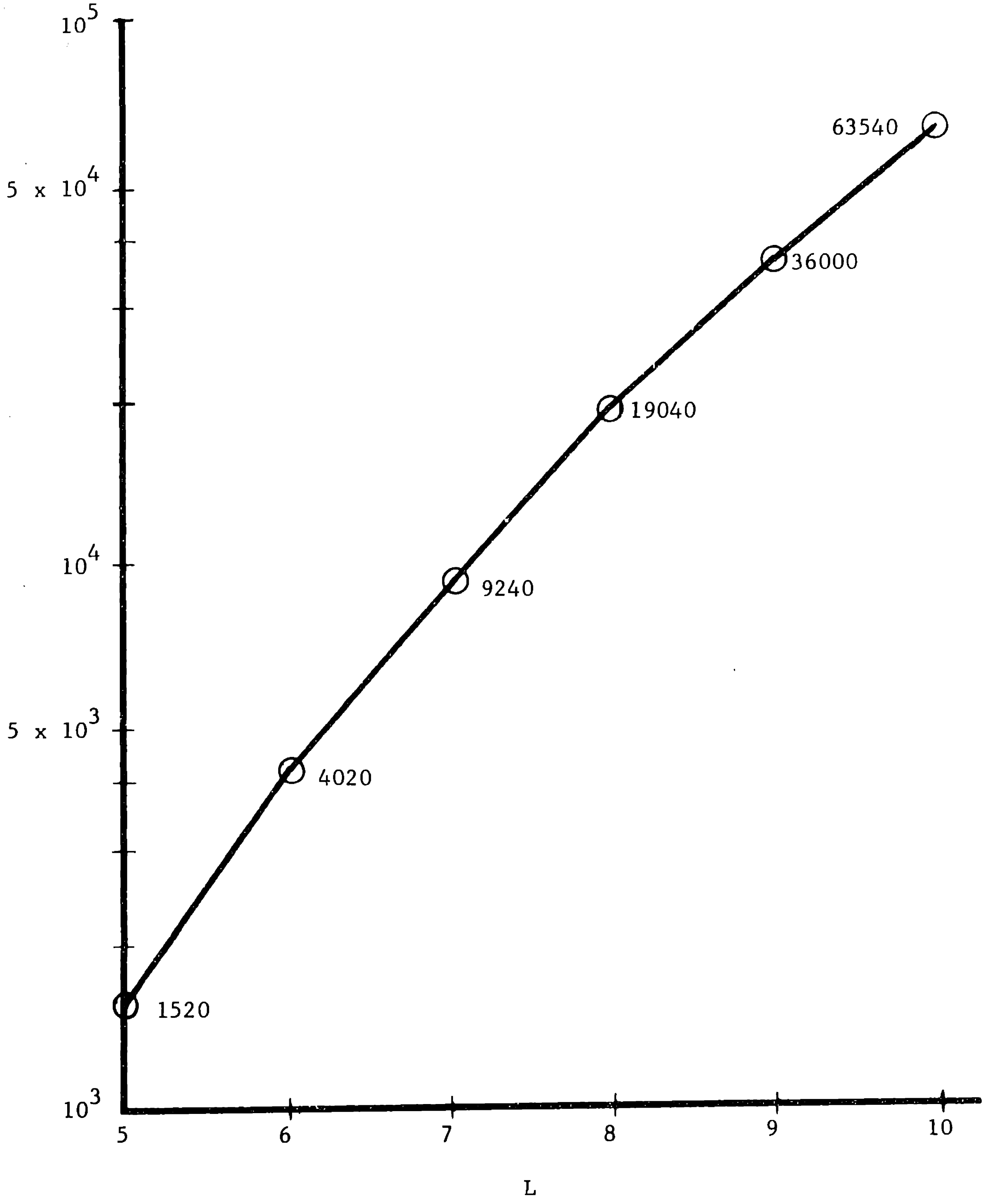

10^5
5×10^4
10^4
5×10^3
10^3
63540
36000
19040
9240
4020
1520
5
6
7
8
9
10
L

Figure 1

same number of elements and if all constants in the dominent relation are paired with themselves. Variables in the dominent relation may be paired with either constants or variables. Thus, the desired list may be formed by picking the next relation in the dominent description (assuming some arbitrary ordering) and comparing it with each inferior relation. Every time a possible match is found, a pair is formed and added to the list of possible relation matches. This continues until all such pairs have been found.

The next operation involves discovering all the allowable matchings between the two descriptions, an allowable match being one where at least one relation is matched, and where no conflicting relations are matched. Two relation matches conflict if they imply two pairings for the same element. Every allowable set of relation matches thus implies a mapping of elements, and so defines a description match. However, by our definition of such a match, it is the relation matches that are to be implied by the mapping. Therefore, a third requirement for an allowable match is that all relation matches implied by the element mapping be included. Another way of stating this is that the element mapping and relation matches defined by an allowable match must be consistent with each other.

Each such match is found only once. The previous procedure found all possible matches, i.e., all possible pairings of objects, and this could involve discovering several matches that are equivalent from the point of view of the relations that are matched.

This operation is implemented by what might be called a "progressive refinement" process. At each step of this process there exist a set of sublists of the original relation pair list. Each of these sublists might be considered a partial match. Each sublist has associated with it an index, which points to some element of the list, and two other lists. The first of these associated lists is the "accepted pair list" which shows all those element pairings implied by the partial match. The second list is the "rejected pair list" which shows all pairings which have been rejected in the formation of this partial match. The initial state of the process consists of the relation pair list with its index pointing to the first relation pair, and the associated lists empty. (An exception to this last condition might be made in those cases where it was desired to allow the program which called the match to specify that certain pairings were to be accepted or rejected a priori.)

At each step of the process each partial match is refined one step. The relation pair pointed to by the index is looked at. If any of the object pairings implied by it conflict with those already on the accepted pair list (two pairings conflict if they imply that an element in one description is paired to two elements in the other) this relation pair must be removed from the partial match. Similarly, if any of the object pairs implied by the relation pair are on the rejected pair list the relation pair must be removed. To remove the relation pair, it is deleted from the list and the index is set to point to the next element in the list.

If the relation pair passes these tests, then it does not conflict with any object pairing decisions made during the formation of the partial match. It will pass these tests if all of its element pairings are either on the accepted pair list or no decision has been made about them. If all pairs implied by matching the relations are already on the accepted pair list, then the relation pair must be included in the match, and this is accomplished by simply setting the index to point to the next relation pair on the list.

In the case where decisions must be made, there will result several partial matches, one for each possible decision. If there are N such undecided element pairs, there must be 2^N new partial matches formed, one for each possible combination of pair acceptances and rejections. The index of each of these partial matches is set to the relation pair following the pair under consideration. In all but one partial match the pair under consideration is deleted, the exception being the one where all the new pairs are accepted. The appropriate additions are made to the accepted and rejected pair lists.

The above process is iterated on all partial matches until the index of each list goes beyond the last entry. At that point all the sublists remaining constitute the set of all allowable matches. To finish the match it is necessary to evaluate the matching function for each of these and choose the highest scoring one.

Since the description above may be somewhat difficult to understand, an example follows. Figure 2(a) gives two descriptions and shows the drawings of which they might be descriptions. Of course, many other descriptions could be given of these drawings and many other drawings could be construed as fitting these descriptions. The drawings are given merely to give some intuitive idea of what the descriptions represent.

Part (b) of the figure shows the initial relation pair list. The asterisk represents the index and is beside the relation pair that is pointed to by the index. The characters "A=" are used to indicate the accepted pair list, and "R=" is used to indicate the rejected pair list.

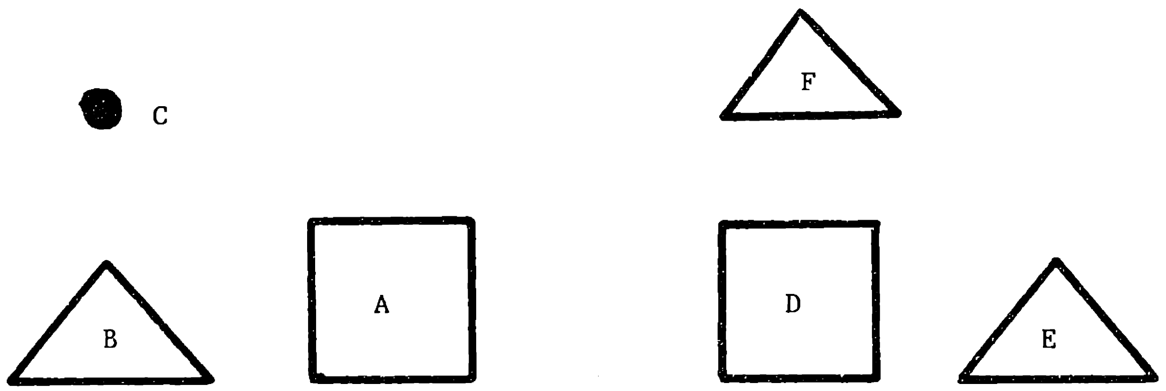
Part (c) shows the partial matches resulting from the first iteration. Since both the accepted and rejected pair lists are empty, decisions must be made about both the pairs (A E) and (B D). Thus there are 2^2 resultant partial matches, one of which accepts both pairs, one of which rejects both, and two of which accept one but not the other.

Part (d) shows the result of the next iteration. The first partial match of part (c) gives rise to the first two partial matches of (d). This is due to the fact that a decision is made as to whether to accept or reject the pair (C F). The pair (B D) had previously been accepted for this partial match. The third partial match of part (d) results from the second one of part (c) upon the rejection of the indexed relation pair because of its conflict with the rejected object pair (B D). The next two partial matches of (d) result from the third match of (c) by a process similar to the first two matches in (d), and the last partial match of (d) follows from the last partial match of (c) by again rejecting the indexed relation pair due to conflict.

Part (e) skips some iterations and in fact does not show the state resulting from a particular iteration. In the interest of brevity all those cases where the rest of the relation pairs in a partial match are rejected due to conflict with previous decisions are shown in their final state, indicated by the absence of an asterisk. In some cases, for example the fifth partial match of part (d), this results in the elimination of the partial match. The third partial match of (e) results from the third match of (d) by accepting the pair (B F); rejecting this pair leads to no match as the pair (A D) conflicts with a previous decision. The last two partial matches of (e) result from the last partial match of (d) by chosing to accept or reject the pair (B E).

Part (f) shows the completed matches. Which one would be picked would of course depend on the evaluation function. If this function valued all relations equally, so that the score was merely the sum of matched relations, then there would be several equally good matches. If the type relations (i.e., SQUARE, TRIANGLE) were valued highly enough, say by adding 5 to the score for each of them matched, versus 1 for all other relations, then the choice would be between pairing (B E) and (A D) or pairing (B F) and (A D). On the other hand, reversing the values gives the pairs (A E) , (B D) and (C F) based upon position. An analogy program, for example, would get two entirely different descriptions of the transformations between these two pictures depending on the matching function chosen.

By making use of knowledge about the matching function being used in addition to using relational information, further efficiencies can sometimes be achieved. For example, this is relevant to the refinement technique when the matching function is such that, no matter what set of relations is initially matched, matching any additional relation will not cause the score to go down. This excludes, for example, the case where there is a relation which must not be matched if the match is to get a non-zero score. Under this condition it



(RIGHT-OF A B)

(ABOVE C B)

(IS B TRIANGLE)

(IS A SQUARE)

(IS C DOT)

Variables = (A, B, C)

(RIGHT-OF E D)

(ABOVE F D)

(IS D SQUARE)

(IS E TRIANGLE)

(IS F TRIANGLE)

Variables = (D, E, F)

(a)

* (RIGHT-OF A B) (RIGHT-OF E D)

(ABOVE C B) (ABOVE F D)

(IS B TRIANGLE) (IS E TRIANGLE)

(IS B TRIANGLE) (IS F TRIANGLE)

(IS A SQUARE) (IS D SQUARE)

A = ()   R = ()

(b)

Figure 2

```
1.  (RIGHT-OF  A  B) (RIGHT-OF  E  D)
   *(ABOVE  C  B) (ABOVE  F  D)
    (IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (A-E, B-D)     R = ()


2. *(ABOVE  C  B) (ABOVE  F  D)
    (IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (A-E)     R = (B-D)


3. *(ABOVE  C  B) (ABOVE  F  D)
    (IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (B-D)     R = (A-E)


4. *(ABOVE  C  B) (ABOVE  F  D)
    (IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = ()     R = (A-E, B-D)
```

Figure 2(c)

```
1.  (RIGHT-OF  A  B) (RIGHT-OF  E  D)
    (ABOVE  C  B) (ABOVE  F  D)
   *(IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (A-E, B-D, C-F)     R = ()


2.  (RIGHT-OF  A  B) (RIGHT-OF  E  D)
   *(IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (A-E, B-D)     R = (C-F)


3. *(IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (A-E)     R = (B-D)


4.  (ABOVE  C  B) (ABOVE  F  D)
   *(IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (B-D, C-F)     R = (A-E)


5. *(IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (B-D)     R = (A-E, C-F)


6. *(IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = ()     R = (A-E, B-D)
```

Figure 2(d)

```
1.  (RIGHT-OF  A  B) (RIGHT-OF  E  D)
    (ABOVE  C  B) (ABOVE  F  D)

    A = (A-E, B-D, C-F)     R = ()


2.  (RIGHT-OF  A  B) (RIGHT-OF  E  D)

    A = (A-E, B-D)     R = (C-F)


3.  (IS  B  TRIANGLE) (IS  F  TRIANGLE)
   *(IS  A  SQUARE) (IS  D  SQUARE)

    A = (A-E, B-F)     R = (B-D)


4.  (ABOVE  C  B) (ABOVE  F  D)

    A = (B-D, C-F)     R = (A-E)


5.  (IS  B  TRIANGLE) (IS  E  TRIANGLE)
   *(IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (B-E)     R = (A-E, B-D)


6. *(IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = ()     R = (A-E, B-D, B-E)
```

Figure 2(e)

```
1.  (RIGHT-OF  A  B) (RIGHT-OF  E  D)
    (ABOVE  C  B) (ABOVE  F  D)

    A = (A-E, B-D, C-F)     R = ()


2.  (RIGHT-OF  A  B) (RIGHT-OF  E  D)

    A = (A-E, B-D)     R = (C-F)


3.  (IS  B  TRIANGLE) (IS  F  TRIANGLE)

    A = (A-E, B-F)     R = (B-D)


4.  (ABOVE  C  B) (ABOVE  F  D)

    A = (B-D, C-F)     R = (A-E)


5.  (IS  B  TRIANGLE) (IS  E  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (B-E, A-D)     R = (A-E, B-D)


6.  (IS  B  TRIANGLE) (IS  E  TRIANGLE)

    A = (B-E)     R = (A-E, B-D, A-D)


7.  (IS  B  TRIANGLE) (IS  F  TRIANGLE)
    (IS  A  SQUARE) (IS  D  SQUARE)

    A = (B-F, A-D)     R = (A-E, B-D, B-E)


8.  (IS  B  TRIANGLE) (IS  F  TRIANGLE)

    A = (B-F)     R = (A-E, B-D, B-E, A-D)


9.  (IS  A  SQUARE) (IS  D  SQUARE)

    A = (A-D)     R = (A-E, B-D, B-E, B-F)
```

Figure 2(f)

is possible to find the best match without finding all matches. To do this the refinement procedure must be modified. Any time a partial match is refined, the scores of all the newly created partial matches are evaluated. In doing this, a relation is considered matched if it has not yet been eliminated from the partial match being evaluated, i.e., if it appears in one or more of the remaining relation pairs. Rather than picking an arbitrary partial match to be refined at each step, the partial match with the highest score is refined. When the condition is achieved that the highest scoring partial match is completely refined (is a completed match), then it is the best match, since all remaining partial matches have lower scores and these scores must stay the same or become smaller as relation pairs are removed, because of the condition stated above. This procedure is similar in concept to the Reinwald-Soland method of generating decision trees [3].

In cases where the matching function is fairly complex, evaluating it at each refinement step can be costly. However, in the case where the function simply counts the number of relations matched, finding the length of the partial matches might serve nearly as well as evaluating the function, even though some relations may be counted more than once.

It is difficult to get a closed form expression that gives a reasonable estimate of the work this algorithm performs as the descriptions get large. This is due to the fact that its operation depends a great deal on tne structure of the descriptions being matched, i.e., how many different types of relations there are, how many instances of each relation occur, and which objects are related to each other in some way. In the example given the refinement technique is definitely more efficient than the other guaranteed method in terms of matches found, as it finds only nine matches while the other method finds 24, many of which are equivalent in terms of which relations are matched, but all of which must be found and evaluated.

Some bounds can be given for the number of matches found. The formula given previously is a good upper bound on the number of matches that will be found by the refinement method, as this number will actually be achieved in cases where only one relation type appears in both descriptions, and every object in both descriptions is related to every other object by that relation (for the formula to apply, the same assumptions about pairing of objects must be used as were stated in its derivation). Thus the refinement method can do no worse, in terms of matches found, than the first technique, and will almost always do better, since descriptions that give the worst-case behavior are not the sort that are encountered in practice.

A lower bound on the number of matches found by the refinement method is

$$\sum_{i=0}^{n} L_i R_i$$

where L_i is the number of instances of relation type i in one description, R_i is the number of instances in the other description, and n is the number of different relation types that appear in the two descriptions. This is just the length of the initial list of all possible relation pairings. Each pair of relations will appear as a singleton match at the end of the refinement process, and so there will be at least as many matches as there are initial relation pairs (an exception occurs when two objects are related by more than one relation, but this makes only a minor difference in most cases). This is not a very good lower bound, however, as it will be achieved only for the simplest descriptions. For descriptions of any size, various combinations of these singleton matches are possible, giving larger matches. These larger matches will normally far outnumber the singleton matches. An estimate can not be made by just finding all combinations of singleton matches, as many of the resulting "matches" would involve conflicting pairings and so would not be matches at all.

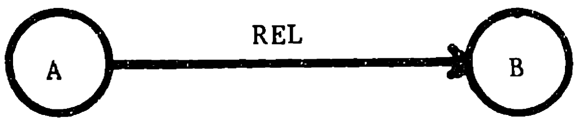
How the number of matches found translates into some measure of computational effort is extremely difficult to see. As a guess, it appears to the author that the refinement method will be the more efficient as the descriptions grow larger, the fact that fewer matches need be looked at eventually compensating for any advantage the other method might have on a per match basis. However it must be emphasized that this is only a conjecture.

Heuristic Methods

The methods described above give guaranteed best matches, but in doing so involve the effort of finding all allowable matches. In some circumstances it might be desirable to find all matches, but often only one is desired. Furthermore, it may be desired to find a match for the dominent description within some small subset of the inferior description, and therefore uneconomical to have to look at every relation in the inferior net. For these reasons, it may be desirable to use heuristic methods which give good matches under most circumstances but which do not always give the best match.

Winston's Matching Program

Winston's matching program is especially designed for descriptions involving only binary relations, which he calls description nets. The discussion below is based on the material in the appendix to his thesis. Some conventions of that thesis make this procedure easier to follow. First, Winston treats his networks as directed graphs, with objects being nodes and relations being represented by labeled directed arrows between these nodes. A relation can be followed in only one direction. Thus in the figure below representing a relation between objects A and B, the match



could find object B if it knew about object A, but could not get to A from B (unless of course it had already passed through A).

The match procedure starts by being given an object from each description. It is important to the correct operation of the match that these objects be likely candidates for pairing. However, they are not paired unless some "evidence" exists that they should be. Evidence for pairing two objects consists of their being related in the same way to some common object or to two objects already paired. In the directed graph representation this means that the nodes representing the two objects are at the tails of two identically labeled arrows pointing to the same node or to two nodes representing objects paired with each other.

Whether or not two nodes are paired, the match then follows the most common relation arrows to find certain of the descendants of the original nodes. An attempt is made to find evidence for pairing any of these. This continues until no more descendents are found. The match then backs up to the previous nodes looked at. If these were not paired the last time they were looked at, a new attempt is made to do so, and will succeed this time if any of their descendents were paired. Whether or not they were paired the last time, the match tries to find any other descendents not previously looked at, such as those which are at the other end of the second most common type of relation arrow coming from those nodes. If no such descendents exist, the match then backs up to these nodes' ancestors. This continues until the match backs up to the original pair of nodes given as arguments.

Again, the operation of this matching procedure is best understood by following through an example. The descriptions to be matched are shown in figure 3 in the node and arrow representation. This example is taken from the above mentioned appendix to Winston's thesis.

Initially, nodes A and A' are given to the match. There is no evidence for pairing them and so nodes C1, C2 , C1' and C2' are looked at next, since the most common relation leaving

A and A' is P. There is evidence for pairing C1 and C1' , but there is more evidence for pairing C2 and C2', as these nodes are identically related to two common nodes. Rather than create both pairs, Winston's match sets up the better of the two. Therefore C2 and C2' are paired.

Looking at the descendents of C2 and C2' the match finds E1, E1' and E2'. There is evidence for pairing E1 with both of the primed nodes. However, E1' is chosen because in addition to being related to a common node it is related to the previously paired node C2' in the same way that E1 is related to C2, the node with which C2' is paired. Therefore there are two items of evidence for paring E1 and E1', versus only one item of evidence for the pair E1-E2'.

Again looking at the descendents of E1 and E1' , evidence is looked for to pair F1 with F1' or F2'. In this case the match uses some knowledge of the evaluation function, to use the terminology of this thesis. It is more important that the MUST-BE-R relation be matched. In the formulation of this thesis, the MUST-BE-SATELLITE relations used by Winston are treated as combining both the relation itself (R in this case) and the MUST-BE indicator which is part of the evaluation function. Using this knowledge, Winston's matching program considers the matching of the MUST-BE-R relation to be better evidence than the matching of the R relation and so pairs F1 and F1'.

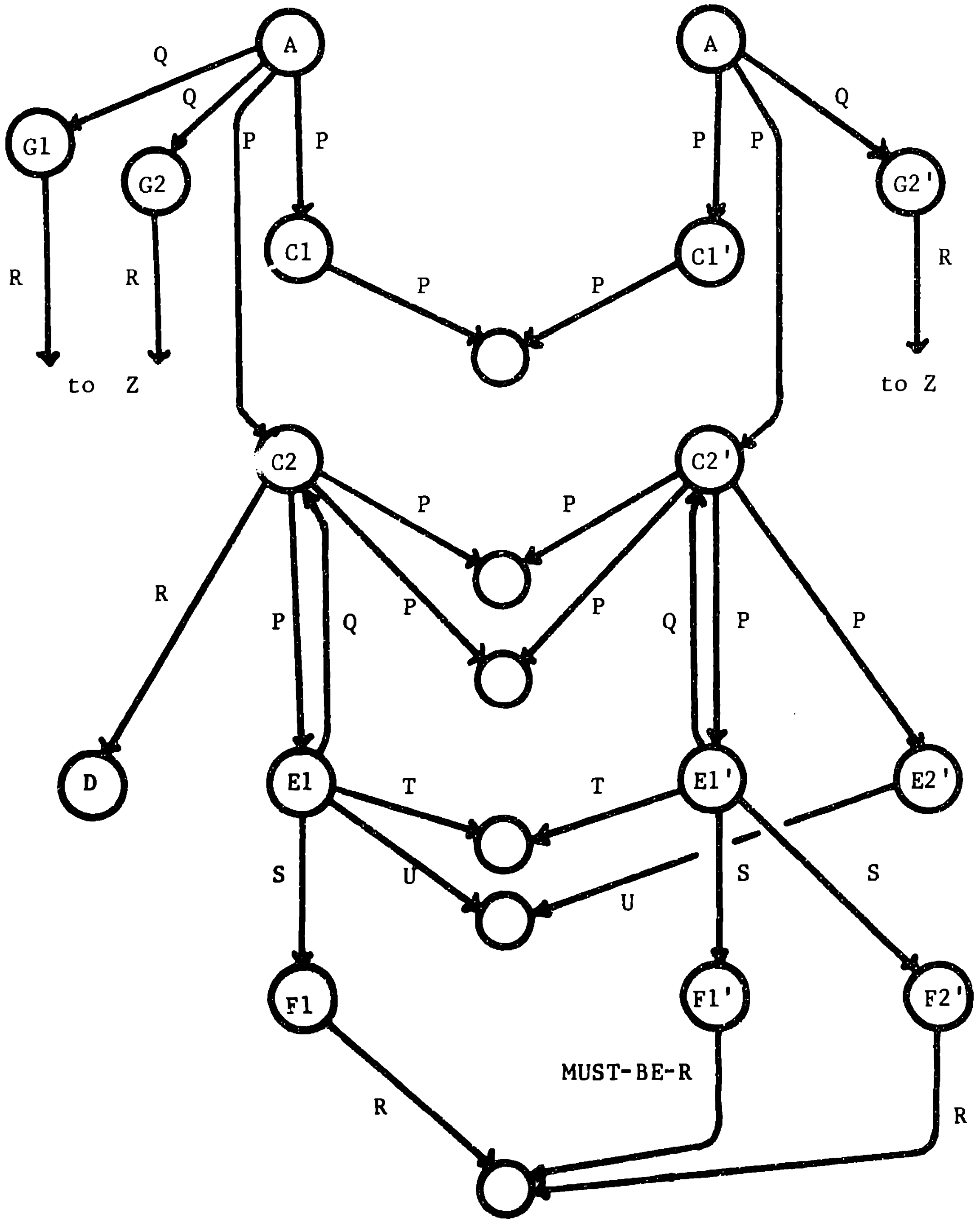

A
A
Q
Q
Q
P
P
P
P
G1
G2
G2'
C1
C1'
R
R
R
to Z
to Z
P
P
C2
C2'
P
P
R
P
Q
P
P
Q
P
P
D
E1
E1'
E2'
T
T
S
U
U
S
S
F1
F1'
F2'
MUST-BE-R
R
R

Figure 3

F1 and F1' have no descendents, and so the match backs up to E1 and E1', and since these are paired and have no further descendents the match again backs up to C2 and C2'. These nodes have unpaired descendents D aND E2', but these are not reachable by identical relations and so no attempt is made to find evidence for pairing them. The match then backs up through C2-C2' to nodes A-A'. These were not paired previously but are at this time because of their common pointers to the paired nodes C2-C2'.

Next the descendents of A-A' are looked at. While P is the most common pointer leaving this pair, if only relations to unpaired nodes are counted, Q is the most common. Therefore G1, G2, and G2' are looked at. Since all have identical pointers to the same node Z , there is not enough evidence to pick one pairing over the other. Therefore G2-G2' is chosen at random.

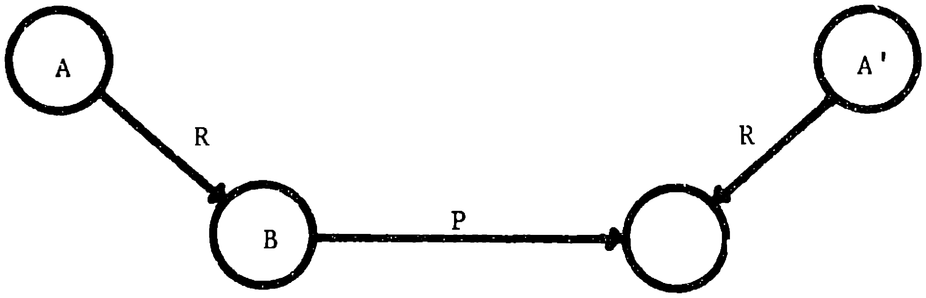
Backing up to A-A' again, since G2 and G2' have no descendents, the match finds that C1 and C1' are the only remaining unpaired descendents. The evidence for pairing them is clear, and the match terminates.

By way of comparison, an attempt was made to hand simulate the refinement method in a matching of these two descriptions. The attempt was given up when it appeared that well over 60 partial matches of fairly large size would result. Thus, on the basis of ease of hand simulation

Winston's method definitely appears to involve less computation, although the price paid is a loss of the guarantee of a best match.

This matching method has several disadvantages. One is its repeated looking ahead and backing up, which results in many nodes being checked several times and much duplication of effort. Another is that objects can only be paired if they are the same "distance" from the original objects handed to the match, in terms of the number of arrows that must be passed over in order to reach them. This is often not serious, however, because there may be several paths of different lengths from the initial node to any particular node of the description.

Another difficulty is that the relations are in fact considered as arrows, to be followed in one direction only. This reduces the number of paths to a node, and so may eliminate a path along which it could be matched. This limitation also restricts the structure of a description and the initial node which may be handed to the match, as the match must be given a node from which it can reach the rest of the description. It would not be possible to get correct matching by handing the match nodes C1 and C1' from the previous example, for instance, even though it appears that a reasonable matching could be defined under those circumstances, and it is conceivable that there might be good reasons for wanting to start with pairing some particular nodes within the descriptions. However, it must be noted that the correct functioning of this matching procedure depends on the relations being one-way arrows, for reasons associated with boundary difficulties. Consider the example below:



The unlabeled node is a common node outside the two descriptions proper but reachable from both. As the match works now, an attempt would be made to pair B and the common node, but this would fail for lack of evidence. Since the common node has no descendents the match would look no further. However, if the match could follow arrows in reverse, node B would be considered part of the description beginning at node A', as would node A after another step. The boundary between descriptions is thus provided by the fact that arrows never go from common nodes which are outside a description into that description.

In our formulation of descriptions, which is somewhat different from Winston's, the boundary is provided by the fact that two descriptions are two entirely separate sets of relations. Although relations to common nodes (constants) certainly exist, these constants are in some sense outside or on the boundary of the descriptions. The fact that constants are recognized as such in our system (i.e., that constants are distiguished from variables) as well as the fact that our descriptions do not connect directly into a larger "description net" makes this possible.

Matching Based on Connectivity

A set of matching techniques based on what might be called the "connectivity" of the descriptions can be formulated. These are based on various local methods (that is, they look at only a small subset of the relations in a description) for deciding which are the best relations to match. The first observation leading to these ideas is that if two objects are paired then the relations in which they are involved will be matched if and only if the other objects in those relations are paired accordingly. These other objects can be considered neighbors of the original objects, as they are only "one relation away." For example, in the following two sets of relations, assume A and A' are paired.

```
(REL1  A  B  C)          (REL1  A'  B'  C')
(REL2  D  A)             (REL2  D'  A')

     (a)                       (b)
```

The two relations in set (a) will be matched by those in set (b) if and only if B is paired with B', C with C', and D with D'. The general idea for a matching program based on this, then, is that whenever two objects are paired, all the relations in which they are involved should be looked at and appropriate pairings made between the other objects in these relations. These new object pairs could be put on a list, and then taken one at a time and their neighbors paired in turn, until the entire description is paired.

There are several difficulties which need to be ironed out before this method can become useful. First, suppose that an object is already paired and yet is a neighbor of a newly paired object. The procedure above would attempt to pair it again, perhaps with a different object. This is fairly easily taken care of, however. If a neighbor of a newly paired object is already paired, then that previous pairing must be kept. If it is not possible to do so and still match the relations involving this previously paired object, then these relations must remain unmatched. In the previous example, for instance, if object D had previously been paired with another object, say E, then the relation (REL2 D A) could not have been matched.

Another more difficult problem is that it is not necessarily desirable to match relations to neighboring objects. This depends upon the evaluation function. In fact, the problem extends somewhat further in that it may not be desirable to pair a neighboring element because that would cause one of the undesirable relations between that element and one of its neighbors to be matched (this includes the previous case, as the neighbor relation is symmetric). These considerations lead to the conclusion that this local matching process must have built into it knowledge of the evaluation function, or at least must be able to use this information, if reasonable results are to be achieved. For some functions, for instance those that give a score to each relation, it is fairly easy to see how a local matching procedure could make use of this information. For some very complex matching functions, however, it may not be possible to determine in any reasonable way how a local decision will affect the final evaluation.

A final difficulty arises because of the possibility that a relation in one description may possibly match several relations in the other. For example , in the following sets of relations, assuming A and A' are the initially paired

```
(REL1  A  B  C)          (REL1  A'  B'  C')
(REL2  D  A)             (REL2  D'  A')
                         (REL2  E'  A')

     (a)                       (b)
```

objects, it is not clear whether to pair D and D' or D and E'. Under many circumstances one or both of these possibilities might be eliminated by some of the considerations above. If they are not, however, some action must be taken.

One possibility is to not pair any of the objects. This has the advantage that one of the possible pairings may later be made when the match follows along some other path through the description. Thus this action postpones a decision in hope that some reason for chosing a particular pairing may turn up later.

Another alternative is to base the choice on more complete knowledge of the evaluation function by making the choice that appears most likely to lead to a high score. This can have the same difficulties discussed above with regard to complication of the evaluation function, but under those circumstances the local match is not likely to be useful anyway. Again, if a simple function is used which simply assigns a value to each relation and adds these together, a good heuristic might be to give a score to each possible set of pairings by adding together the values of the newly matched relations resulting from that set of pairings.

A final alternative would be to make an arbitrary choice. In order to provide some way to repair bad choices made in this way some backup mechanism might be used. This could lead to a system where all possible choices were tried in turn.

This would come about by using an arbitrary choice mechanism until the match was completed, evaluating the resultant match, then backing up to the last arbitrary choice and taking a different decision. This new match would also be completed and evaluated. This process would continue until all possible alternatives were looked at. This requires some mechanism for keeping track of possible choices and which of these have been used, and could prove quite expensive.

Using Uncommon Relations

The local matching method above assumes that it will be given one or more pairs of objects from each description, these objects to be paired a priori. This initial pairing might be provided by the programs which call on the matching procedure. However, under certain circumstances it may be desirable for the matching procedures themselves to provide a good initial pairing. One heuristic which can prove useful for doing this under certain circumstances involves searching the two descriptions to find those types of relation which occur least often. By some means one such relation from each descripion is chosen and this pair of relations is matched, resulting in the pairing of the objects involved in those relations. This pairing of objects then provides an initial pairing for the local match (which can, of course, be given more than one node pair initially). The considerations involved in methods for choosing the appropriate relations to match are similar to those mentioned above for the case when the local match has a choice to make.

This heuristic can also be useful after the local match has terminated. If, for example, the local match sometimes does not choose between alternative pairings, preferring to wait to see if one of the pairs will be chosen at a later stage of the match, a situation can arise in which the local match runs out of objects to pair, and yet a large portion of the description remains which could be matched. An example will be given below. Under these circumstances it might be desirable to look some distance into the unmatched portion of the descriptions and find likely pairs of objects by use of the uncommon relation heuristic. The local match method could then be restarted using these new pairings.

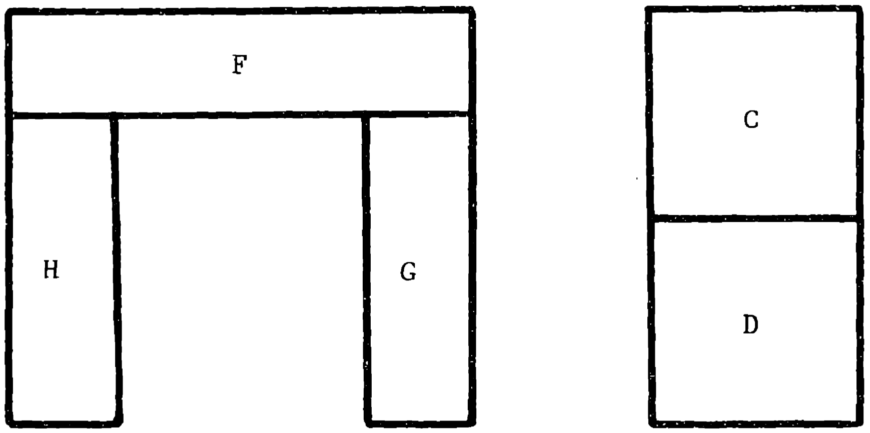
An Example

In this section an example is given of the use of a local match technique combined with the uncommon relation technique. The evaluation function used is simply based on a count of the number of relations matched, thus giving each relation equal importance. The descriptions being matched contain only binary relations, and so are represented by a network. Figure 4 shows the networks and the scene they are intended to be representations of. Both networks represent the same scene, shown in part (a) of the figure. The difference is that in one description (part (b)) the rectangles have been recognized as an arch and described as such, whereas in the description in part (c) this grouping has not been made.

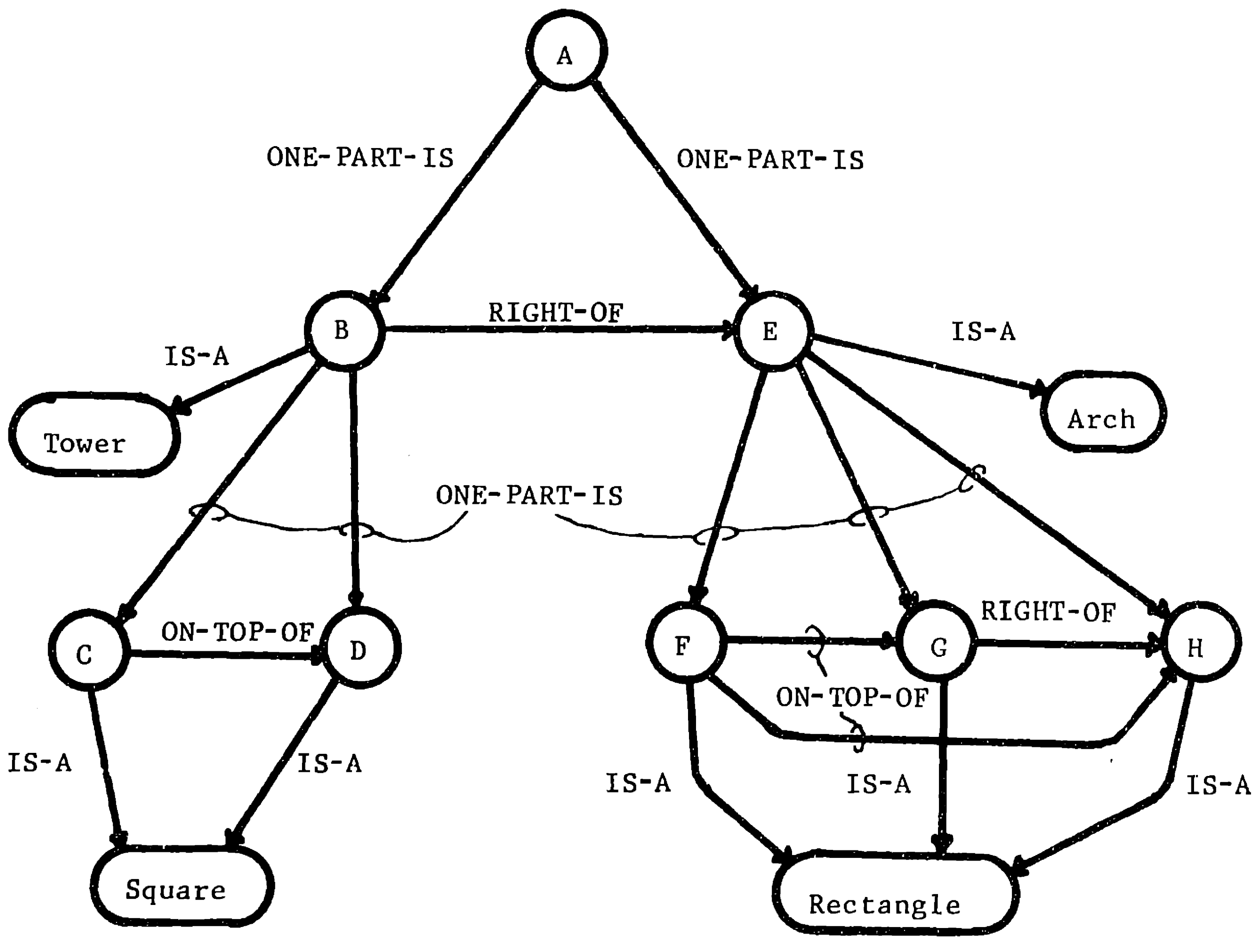
Initially the local matching process is given nodes A and A' to pair. All the relations leaving these two nodes are identical, and so all the neighboring nodes are candidates for pairing. Thus some decision must be made as to which pairings are best. Since the evaluation function scores on the basis of the number of relations matched, a good heuristic is to evaluate each pairing and pick those giving the greatest number of matches. Actually, the number of potential matches is what will be counted, including in the count all relations which might eventually be matched. All the possible pairings are checked, and give the following potential match counts:

B-B'	B-F'	B-G'	B-H'	E-B'	E-F'	E-G'	E-H'
5	1	2	1	3	2	2	2

Starting with the highest count, then, B would be paired with B'. This eliminates all but three of the pairs, and all

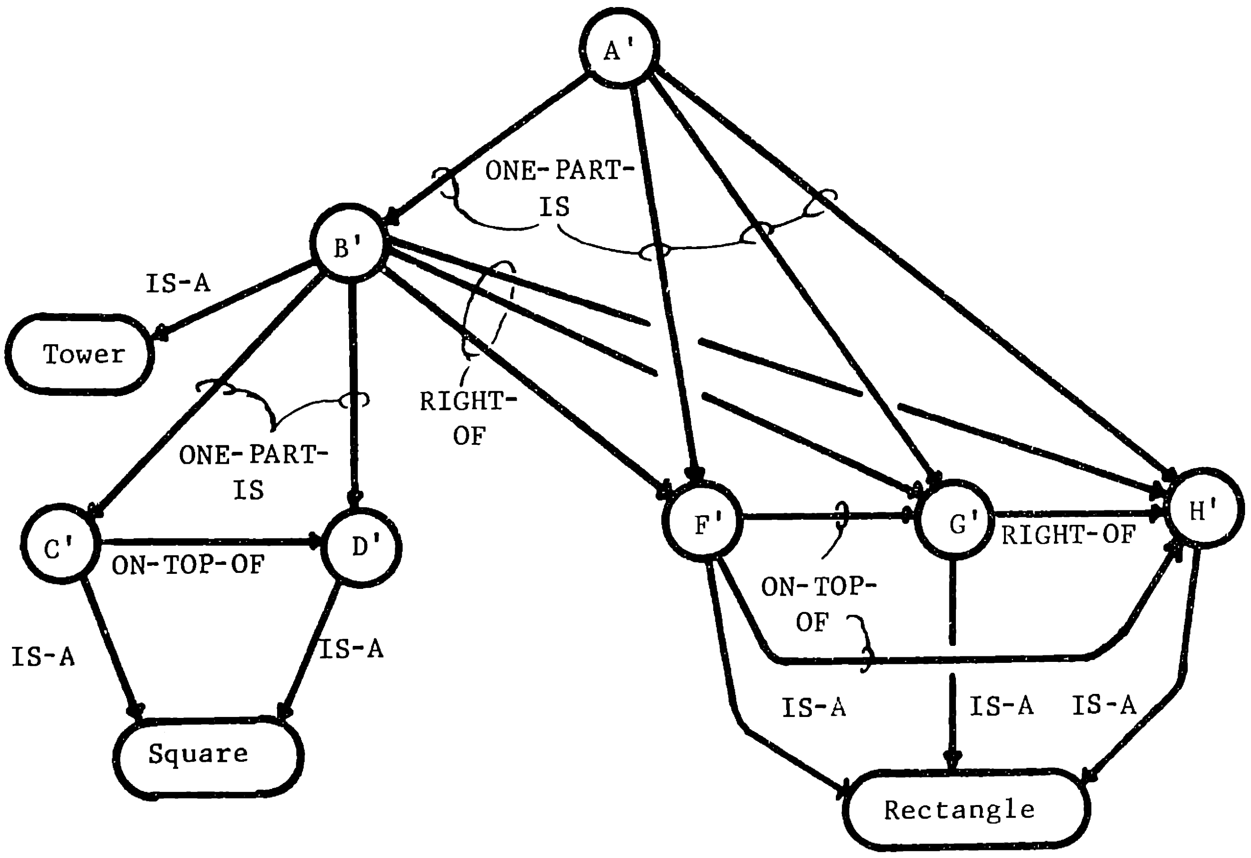


(a) The Scene



(b) One Description

Figure 4



(c) An Alternative Description

Figure 4 (continued)

of these have identical counts. Under these circumstances the local match gives up for lack of evidence, and so E, F', G', and H' are not paired. Since the only new pair formed was B-B', the neighbors of these two nodes are looked at next. The RIGHT-OF relations again cause an attempt at pairing E with F', G' , and H', and this fails for the same reason. The ONE-PART-IS relations cause evaluation of the pairs C-C', C-D', D-C', and D-D', the counts being as follows:

C-C'	C-D'	D-C'	D-D'
3	2	2	3

Therefore nodes C and C', D and D' are paired. At this point the local match terminates, as these newly paired nodes have no unpaired neighbors. However, there are still unpaired nodes in the descriptions and so an attempt is made to locate the least common unmatched relations. Which relations are least common is determined by multiplying the number of instances of a particular relation in one description by the number of instances in the other and comparing the resulting numbers. Thus the comparison is based on the number of possible matches suggested by that relation type. Using this criterion, ON-TOP-OF is the least common relation, suggesting only four possible relation matches.

However, this uncommon relation routine is intended to return a list of node pairs to the local match. One way it could do this would be to try to find the best match or matches among the relations found and return the pairs implied by those matches. A simpler technique, however, is to score all the pairings suggested by these relations in the same way used in the local match and to return those pairs with the highest score. The scores for the pairs suggested by the ON-TOP-OF relations are:

F-F'	G-G'	G-H'	H-G'	H-H'
3	3	2	2	3

Notice that the ONE-PART-IS relations coming into these nodes cannot be counted, as node A' is matched with node A and not node E. Similarly, the RIGHT-OF relation from node B' cannot be counted as matching the RIGHT-OF relation originating on node G.

On the basis of the above scores pairs F-F', G-G', and H-H' would be returned. The local match could do no more, and the matching process would be completed. Under certain circumstances it can happen that several of the top scoring node pairs in the uncommon relation routine conflict with one another, in that more than one pairing is suggested for a node. Under these circumstances either all the conflicting pairs can be rejected or else one of them picked at random and the others rejected.

In looking at the results of this match several observations can be made. First, it is not a best match according to the evaluation function used. In the best match node A' is paired with node E. Secondly, the match obtained may really be the desirable one, since it brings out the missing feature in the second description, the lack of a node to represent the arch. This is a case where finding the best match according to the evaluation function may not correspond to the best match in terms of what seems intuitively best, and points up some of the difficulties of description matching of the sort being discussed here.

Match Improvement

The heuristic methods above do not always give the best match based on the evaluation function, as demonstrated in the example above. It appears as though it might be desirable to try to improve on these initial heuristic matches where possible (although the comments in the last paragraph of the previous section indicate that this is not necessarily so). This leads to two questions. First, is it possible to cheaply determine when improvements can be made? Second, how can a better match be found based on the initial heuristic match?

It is always possible to answer both questions by finding the best match using some exhaustive procedure. Doing this, however, negates the value of any information the initial match might provide. In the case of the particularly simple evaluation function which just counts the number of relations matched, there is a technique for telling when further improvement is impossible. If there are no relations of the same type left unmatched in both descriptions, then further improvement is impossible. This simply indicates that any unmatched relation cannot be matched without forcing another relation to become unmatched. Since they are both valued equally no increase in score is possible. If this condition does not hold it says nothing about whether improvement is or is not possible.

In the case of more general evaluation functions it is hard to say anything applicable to all of them. Determining whether improvement is possible and methods for achieving this improvement are probably very dependent on the evaluation function and on the particular purpose the descriptions are being used for. Knowledge of this context of usage could lead to the design of various techniques applicable to special cases.

Conclusions

This chapter has presented several methods of matching descriptions in order to obtain a comparison of them. The methods have not been presented in great detail and some fine points have been ignored. This is due to the fact that to a large extent all these techniques can be combined and modified in various ways. What has been presented is fairly generally applicable, and details can be filled in to suit the application. In any situation where it appears that use of some sort of internal descriptions with comparison might prove valuable, it is first necessary to determine if the particular sort of description discussed here appears useful and, if so, it must then be decided just what the detailed structure of the description mechanism will be, including the matching mechanism. In any system involving large descriptions and complex description learning it could take considerable experimentation to adjust these mechanisms to get the desired results. The difficulty here is that this particular form of description does not in and of itself define one or more "efficient" or "optimal" ways of organizing descriptions and comparing them.

In the next chapter a particular description matching program will be discussed. This program was actually implemented and some experiments performed with it.

V. Some Experiments

A description matching program designed along the lines of the heuristic programs mentioned in the last chapter was programmed in LISP and Micro-Planner. The objective was to design a matching program that would match descriptions similar to those used by Winston in his learning program. It was hoped that by using some of these heuristic methods a better matching program than the one used by Winston would result, "better" being taken to mean both that its operation would be more efficient and that it would be able to do a better job of matching descriptions which Winston's matcher could not handle. The second objective was achieved. The objective of efficiency was found to be very hard to evaluate, and while some comments will be made about it, no definite comparison of the two methods on that basis will be made.

The Descriptions

Relations in these descriptions were restricted to being binary relations. Furthermore, relation names were not allowed to be variables. Use was made of Micro-Planner's data base facilities to store the relations as Planner assertions.

The evaluation functions were not represented explicitly by programs, but were instead implicitly used in the matching procedures. All non-emphatic relations were weighted equally, providing a simple basis for evaluation. This seems necessary if the heuristic match procedures are to be able to get a reasonable idea of the best object pairings to make on the basis of local information only. Emphatic relations were handled by causing the match to fail (receive a zero score) if a MUST-BE satellite relation was not matched or if a MUST-NOT-BE satellite was. This definition of MUST-NOT-BE runs into the problem mentioned in chapter three that under no circumstances will a MUST-NOT-BE relation ever be matched. However, it still proves useful, for in not matching this negative emphatic the match may be forced into not matching a MUST-BE emphatic, and so can still recognize certain cases where an instance does not fit a concept. A simple example is given below. In order for an instance to satisfy this

```
(MUST-BE-ABOVE  A  B)
(IS  A  CUBE)
(IS  B  CUBE)
(MUST-NOT-BE-GREEN  A)

Variables = (A, B)
```

concept, A must be above B and A cannot be colored green (It is not necessary that A and B be cubes, as the IS relations are not emphatic). If this concept is matched against a description where one cube is above another and the top one is green, the matching procedures will refuse to match the top object with A. This prevents the MUST-BE-ABOVE relation from being satisfied, however, and so the match will receive a zero score anyhow.

Overview of the Matching Procedures

The matching procedures used were a combination of a match using connectivity and a match using uncommon relations. In addition, a match improver was programmed. All of these made extensive use of "pair evaluation," a means of scoring possible object pairings that was like the count of potential relation matches mentioned in an example in the previous chapter. At the top level, the matching program was called with a list of initial object pairings. The connectivity match, which was called the local match because it used only local information in its pairing decisions, was then applied to these initial object pairings, pairing their neighbors, then the neighbors of the newly paired objects, and so on until no more neighbors of paired objects could be paired. During this process, it was possible that the local match would refuse to make a choice among possible pairings, and as mentioned previously this occasionally led to unmatched portions of a description. Whenever this occured an uncommon relations method, called the extended match, was applied. This looked some distance (specified by an argument to the match procedures) into the unmatched portion of the description and on the basis of the least common relation type would recommend one or more object pairings. These were then given to the local match, and it paired some neighborhood of these objects. This process continued until no more objects could be paired.

At this point an initial match was said to have been achieved. A match improvement program was then called. It first looked to see if there might be some possibility of improving the initial match. If it did appear that this might be possible an attempt was made to do so. This was done by choosing some alternative pair of objects, a pairing which did not appear in the initial match, and using this to start a new match using the local match program. The extended match was not used in forming this alternative match. If the alternative match received a higher score than the initial match, it replaced the initial match and an attempt was made to improve it. If it received the same or lower score, it was discarded and a new alternative initial pairing chosen. This process halted when no more plausible alternative pairings could be found or when it was shown that the match could not be improved further.

These procedures will be discussed in more detail below. First the pair evaluation mechanism will be described, followed by descriptions of the local and extended matchers. Finally, the match improver will be discussed.

Pair Evaluation

The object of pair evaluation is to get some approximate idea of how much the match will be improved by pairing two particular objects. With the simple evaluation function of the descriptions used in these experiments this is roughly accomplished by adding up the number of matched relations that will be added to the match if these objects are paired. This statement requires some modification, however.

First of all, potential rather than actual relation matches are counted. In order to do this, all relations involving the two objects in question are looked at. In order that a relation in one description potentially match another, it must be true that

1) The relation names are the same

2) a) The other objects in the relation are unpaired

or

b) The other objects in the relations are paired to each other.

2(a) makes the assumption that the presently unpaired objects will be paired to each other at some later stage of the match.

Second, account must be taken of emphatic relations. In order to do this, all MUST-BE relations are looked at first, and an attempt made to find potential matches for them. If none are found a score of zero for the pair results. If such potential matches are found they are counted the same as matches of non-emphatic relations. MUST-NOT-BE relations are treated somewhat differently, as only actual and not potential matches of these are counted. If pairing two objects would force the matching of a MUST-NOT-BE relation, the pair is given a zero score.

The Local Match

Having a description of the pair evaluation mechanism, the local match is fairly easily described. A list, the "new pair list," is maintained. It initially contains those object pairings given to the local match as arguments. Pairs are taken from the head of this list, and all the neighbors of the paired objects are found. All pairings among these neighbors are computed, and each such pairing given to the pair evaluator. This results in a list of pairs and the scores for those pairs. The highest scoring pair(s) are accepted as new object pairings, and all conflicting pairings (those involving any of the newly paired objects) are removed from the list. Then the highest scoring remaining pairs are accepted, this process continuing until the list is empty or until all remaining scores are zero. All these new object pairs are added to the end of the new pair list.

One modification needs to be made to the above. Occaisionally two conflicting pairs will receive the same score. Under these circumstances the local match does not attempt to make a decision, but rather places all the unpaired objects involved on an argument list for the extended match. This list is passed to the extended match when the new pair list becomes empty, forcing the local match to stop.

The Extended Match

The extended match is called with the argument list created by the local match and a maximum distance limit which says how far into the descriptions it can look. It takes the objects on the list, finds their neighbors, then their neighbor's neighbors, and so on until the distance limit is reached or until no new objects are found. All relations involving the newly found objects are then looked at, and that relation type which occurs least frequently in the two descriptions is determined. "Least frequently" is defined in terms of minimizing the product of the number of instances of a relation type in each description. Every possible match of relations of this type implies a pairing of objects. All such object pairings are collected and pair evaluated. The highest scoring pairs are returned to the local match if they don't conflict. Otherwise one of the highest scoring pairs is chosen at random.

The Match Improver

The match improver is called when the local and extended matches have run out of things to do. It first attempts to discover if improvement is impossible, using the method mentioned briefly in the previous chapter. It finds all the unmatched relations remaining in both descriptions. If none of the unmatched relations in either description are of the same type as any unmatched relations in the other description, improvement is impossible. Under these circumstances matching one of the unmatched relations will merely force the unmatching of at least one currently matched relation, and this cannot lead to improvement of the overall match.

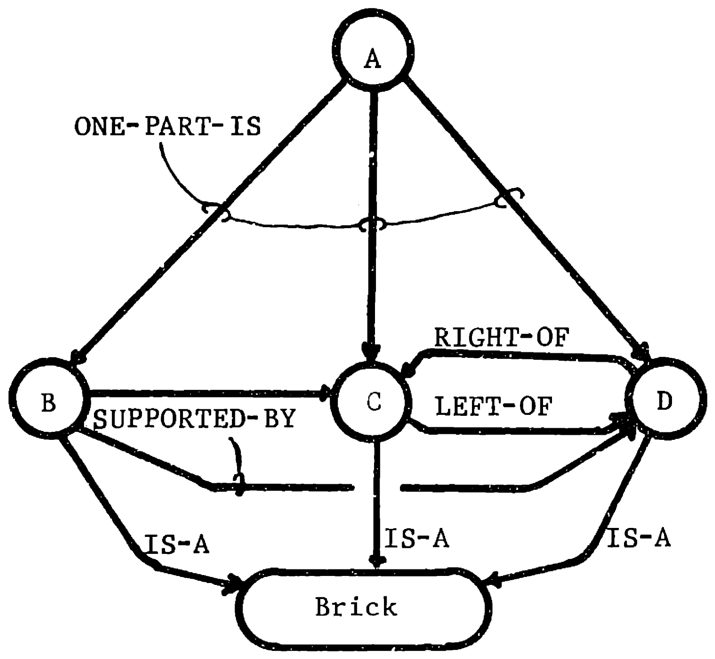
If it is determined that improvement of the match may be possible, a new initial pair for the local match is chosen. This is done by using the uncommon relation method on the unmatched relations, and pair evaluating the suggested pairs, much as in the extended match. This is a heuristic which tends to assume that unmatched relations will be matched to each other, which is not necessarily so. The highest scoring pair is given to the local match (the other pairs being saved) and a local matching of its neighbors formed. This new match is combined with the old match, all pairs of the old match which conflict with new pairings being discarded and all non-conflicting pairs being retained. This new match is then evaluated. If its score is greater than that of the old match, it replaces the old match and an attempt is made to improve it. Otherwise it is discarded, and another pair is chosen from the list of possible alternative pairings. Match improvement stops when further improvement is impossible or the list of alternative pairings empties.

Results of Usage

The matching program described above was applied to several description matches, some of the descriptions being similar in structure to those used by Winston and others having a somewhat different structure. The results can be discussed along two dimensions, these being the quality of the matches procuced, and how efficient are these procedures at producing these matches.

The quality of the matches produced was quite good. Matching was tried on descriptions that were quite similar and also on those that had differences in structure similar to the absence of the arch grouping node in the example from the previous chapter. In both cases the best match was found, even though in the case of the differently structured descriptions this involved the use of the match improver to unpair the initial pair given to the match and re-pair these objects. This is one operation that the matcher used by Winston could not perform. The ability to look into the description and pair objects that are different distances from the initial object pair is another operation not feasible in Winston's program but which can be performed by the extended match.

An example showing both these features is given in figure 1, using the node and arrow representation. If given nodes A and E as the initial pairing, the initial match will result in the pairs A-E, B-H, C-I, and D-J. The match improver will change the pair A-E to A-G, thus finding the instance of the left description in the right description. Winston's program, having only the IS-A relations to BRICK as evidence, would be able to produce a partial match, but not as good a one as even


A
ONE-PART-IS
RIGHT-OF
B
C
D
LEFT-OF
SUPPORTED-BY
IS-A
IS-A
IS-A
Brick

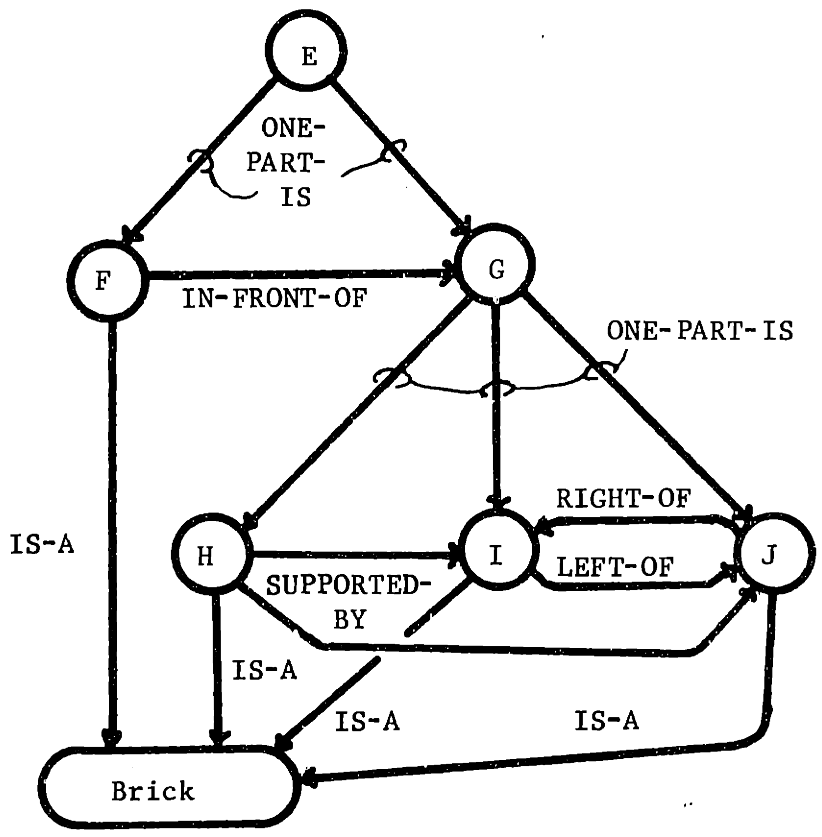

E
ONE-PART-IS
F
G
IN-FRONT-OF
ONE-PART-IS
RIGHT-OF
IS-A
H
I
J
LEFT-OF
SUPPORTED-BY
IS-A
IS-A
IS-A
Brick

Figure 1

the initial match given by these new procedures. Thus these procedures seem to give definite improvements in the complexity of match that is possible. These procedures were also tried out on very large descriptions and gave good results, at least when the descriptions matched were similar.

The efficiency of these procedures is less clear. Since these procedures have no primitive procedure in common with Winston's program, it appears impossible to compare them in any implementation independent way. Since they were implemented by a different programmer using different data structures and representations, it would be fruitless to compare timings with Winston's program. Therefore it appears that no worthwhile comparison with Winston's program can be made in terms of efficiency.

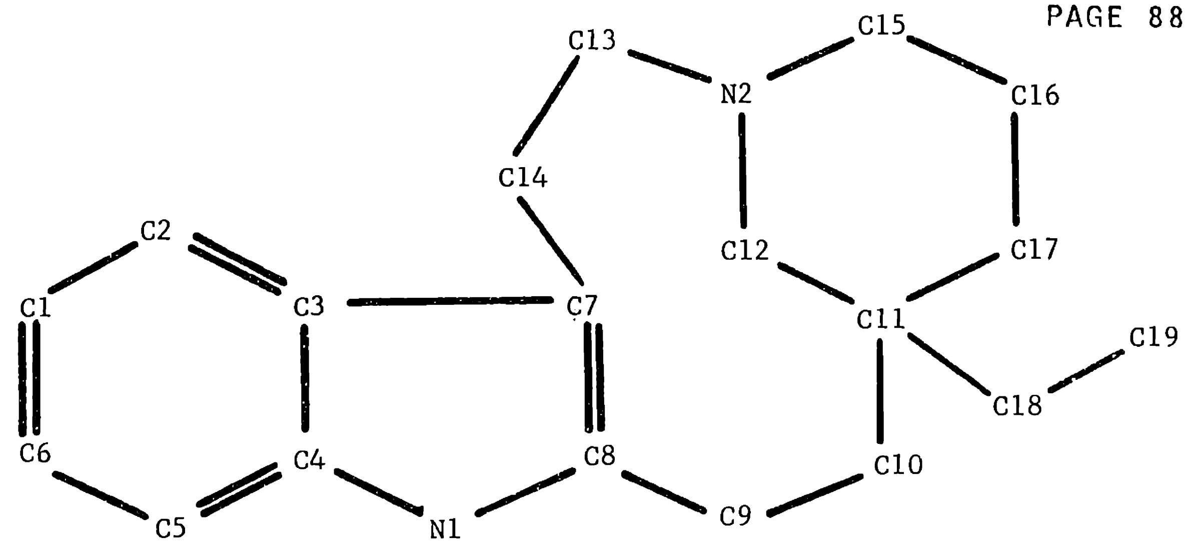
However, some interesting behavior with regard to the efficient running of this program under different situations was observed, particularly when large descriptions were being matched. First, it was found in at least one case that the quality of the initial match depended on how deep into the net the extended match procedure was allowed to look, deeper searches giving better results. In the particular case involved this was due to the fact that by looking further the extended match was able to find a rare relation that was useful in indicating the correct pairs to match. Without such an extensive search, more common relations were used as indicators and often led to incorrect pairings. The resulting initial match tended to be quite bad under these circumstances.

When a bad initial match such as described above was passed to the match improver, it resulted in a great deal of thrashing and little improvement. The reason for this was that in such cases there were a large number of unmatched relations, leading to a large list of suggested alternative pairings. However, it was true that because of the way this list was formed some good pairings did appear at the head of this list. A more important problem was the fact that when given these object pairs the local match was not able to rematch a significant portion of the descriptions. Instead it was stopped due to conflicting pairings with equal scores. As a result, the evaluation score was improved little if at all, and so the program would most often start trying other incorrect initial pairings.

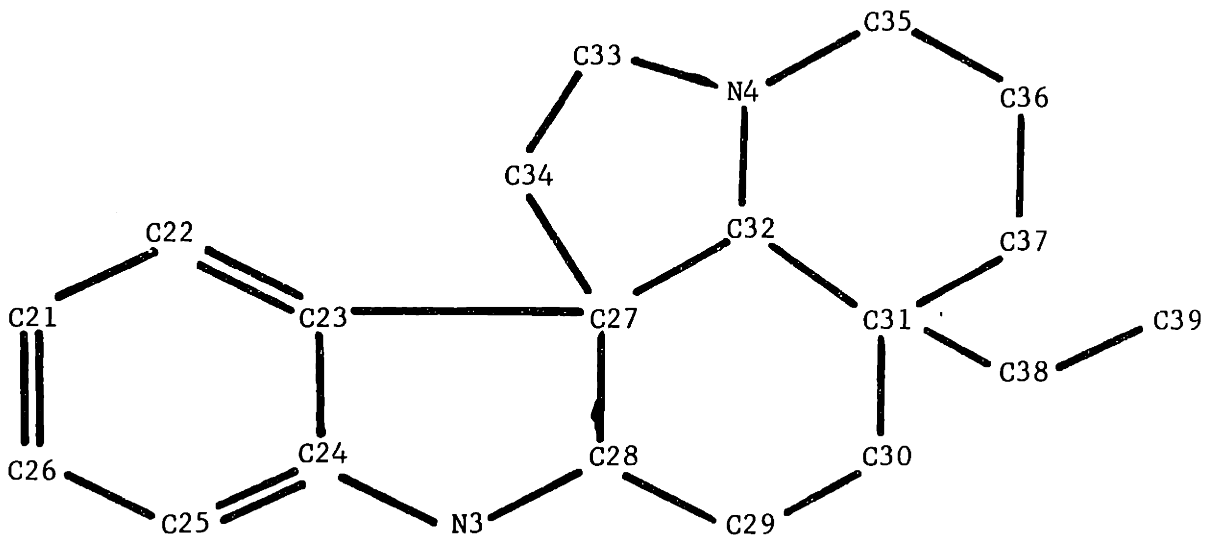
An example where this was particularly bad is shown in Figure 2. The descriptions represent two large organic molecules which are very similar, and they are shown in the usual atom and bond format used by chemists, rather than as relational descriptions. The relational representation used corresponded closely to the diagram, atoms being represented as objects and bonds by relations between objects. For example, (S C1 C2) was used to represent a single bond between atoms C1 and C2, and (IS C1 CARBON) was used to show that C1 was a carbon atom. As is usual in organic chemical representations, the hydrogen atoms were left out of both the figure and the relational description.

Note that almost all the bonds (relations) are of the same type. Similarly, most of the atoms are carbons . The nitrogen atoms (those atoms whose first letter is N) are the best indicators of the correct matching of certain portions of the descriptions. However, the depth to which the extended match was initially set to look prevented it from finding one of these nitrogens at a crucial point in the match, and the situation described above developed. The fact that almost all relations were identical and that a large number of them were unmatched caused a great deal of thrashing on the part of the match improver, as it tried matching all the unmatched carbon-carbon single bonds with each other. It finally halted, having almost but not quite produced a best match. However, when the extended match was allowed to look farther into the two descriptions, the resulting initial match was almost as good as the final match in the previous case, and the match improver easily completed a best match. All this was done in a small fraction of the time required when the extended match was restricted in its range.

The descriptions of large molecules used in the test above were particularly bad for these match procedures, as



Quebrachamine



Aspidospermine

Figure 2

they were very large, and most relations were of the same type. However, the results may give some indication of the failings of these procedures when applied to large descriptions. The local match tends to be unable to proceed very far on its own before being stopped by its refusal to choose between conflicting pairs. The extended match appears to perform very well in helping out the local match, but only if it is able to find a very indicative, rare relationship in the part of the description it looks at. The match improver works well when the initial match is close to a best match, but thrashes and gives bad final matches otherwise. Since it is a fairly large and costly program, it is clearly not always worthwhile.

VI. Conclusions and Further Work

In this thesis a particular form of description is presented and a definition of description comparison given for this description type. While this form of description is not powerful enough to support all the uses that might be made of descriptions and description comparison, it does seem able to support interesting behavior, judging by the uses that have been made of similar forms of description. More importantly, discussion of this particular form has brought out several points which seem likely to apply to any descriptive mechanism.

First, while it does appear desirable to be able to directly express relations among objects, more structure than this is needed for a general descriptive mechanism. Even for the simple descriptions discussed in this thesis it was necessary to introduce variables and an evaluation function in order to represent concept descriptions. Furthermore, the fact that the matching procedures can be given a list of initial object pairings hints that these concept descriptions are acting something like subroutines with the initial pairings as argumant lists. In this analogy, the description whose evaluation function is being used (the dominent description) acts like a program which is interpreted by the matching procedures. Its arguments are a list of initial object pairings, which assign values to certain of its variables, and during the "execution" of the description certain other variables receive assignments (pairings). The results returned are a list of variable assignments and perhaps the value of the evaluation function or a list of matched or unmatched relations. Descriptions differ from the usual idea of programs in that they alone do not completely specify a procedure for assigning values to variables. They merely specify the goal of maximizing an evaluation function given certain relationships. The rest is specified in the matching functions which "interpret" the description. It was mentioned that a complete descriptive ability requires the power of what we have called a recognition process, which involves changes to the data base and transfers of control. All of this tends toward an idea of thinking of descriptions in terms of programs or procedures, incorporating all the structures which are allowed in programming languages.

This idea has been dealt with in Hewitt's Planner language. This language incorporates a form of description comparison and implements several ideas about goal oriented as opposed to procedural oriented programming. However, only a very simple idea of description comparison is built into that language. This comparison is something like the relation matching used in this thesis, but with the evaluation function restricted to a binary function. An instance either fits a concept or it does not, under this sort of evaluation. Also no information is obtained about the differences between two descriptions in Planner's comparison operation, other than that differences exist or they do not (of course, since Planner is a complete programming language more complex comparisons could be programmed into it).

Next is the issue of context. It was hoped that an idea of description comparison could be formulated which was general in the sense that it would be equally applicable to a wide variety of situations and would not depend on knowledge about the area of application. This has been achieved in a limited sense. However, it appears that this may not be the right approach, and that perhaps a better way to get "generality" is to learn how to get many specialized procedures to cooperate rather than to have one "general" technique for all situations.

In the comparison mechanisms discussed in this thesis the information about context of application is hidden in the evaluation function. This is perhaps not too bad, as it seems necessary to have something of this sort to specify how important the various relationships are and how they interact with one another. Some such structure must be present in any description comparison operation, if only implicitly, as there are in general many possible ways of looking at the "differences" between two descriptions. This was demonstrated in a simple way by the example in the introduction to this thesis. Some way must be provided to decide the "best" way to compare the two descriptions, and this "best" way will in general depend on what differences are most important to the particular application at hand.

The actual matching procedures are intended to be context free, and to a large extent they are. The guaranteed methods certainly are, as they merely find all allowable matches and let the evaluation function make the final choice. The heuristic methods, because they use some knowledge of the evaluation function in order to provide a reasonably good match on the first attempt, are less context free. In fact it will often prove advantageous and perhaps necessary to program knowledge of the class of evaluation functions being used into these procedures, as was done for the prodedures described in the chapter on experiments (this requires that all evaluation functions be of the same general type). It is difficult to see how a match procedure could be programmed to make effective use of arbitrary evaluation functions in making local heuristic decisions. However, the general ideas of using the connectivity and uncommon relations of a description are applicable in a wide variety of situations.

How would these relatively general methods compare with specialized methods designed especially for the application at hand? Suppose descriptions were written like executable programs and each description could have built into it any heuristics that might help it to obtain a match quickly. The ability to compare such concept descriptions with each other might be lost due to the difficulty of comparing programs, but it seems that the efficiency with which these descriptions could be "compared" to ("applied to" might be better terminology under these circumstances) a description of, say, a scene, would be increased greatly. For example, in the last chapter it was mentioned that the matching procedures disscussed there had difficulty with certain descriptions of large organic molecules. These descriptions were fairly large and most bonds were carbon-carbon single bonds. Thus the few nitrogen atoms tended to be indicative of good matches. The extended match did find these after some searching and computation of the number of occurances of all the various relations found, but in the general area of comparing such descriptions of organic molecules it would be a good heuristic to search specifically for nitrogen atoms (or all atom types other than carbon and hydrogen).

Further Work

As can perhaps be gathered from the preceeding paragraphs, I would not suggest that anyone pursue the general sorts of description comparison discussed here. However, I do feel that descriptions and especially the ability to compare descriptions are important ideas that should be pursued; it is just that they are too complex and too little understood for much to be done by general methods, at least at this point in time. Useful work could be done by trying to find just how complex a task the simple sort of description mentioned here can support, perhaps by trying to teach a learning program such as Winston's more complex concepts. New descriptive formats should be developed. It would be interesting to see if more powerful descriptive methods, tending toward the representation of descriptions as programs, can be developed which still can be compared in an effective way. This can lead into considerations of how a computer can be programmed to program itself. This would come about when attempts were made to write programs which could learn concepts, these concepts being internally represented by program/descriptions. During the learning process the program would have to construct and modify these descriptions, and in doing so would have to make use of some description programming language and knowledge of programming techniques in that language.

The best way to pursue research along these lines, and this seem to hold true in most areas of artificial intelligence research given the present state of knowledge in this area, is to pick a specific task for a program to perform. The task should be chosen in such a way that it appears to require the descriptive mechanism to be studied. Also, it should be such that it appears possible to write a program, using that mechanism, which will actually perform the task. An alternative approach advocated by some is to pick tasks to perform which seem "do-able" and to discover just what mechanisms are needed for their performance. In this way it is hoped that some understanding will be built up of the mechanisms required for intelligent behavior.

BIBLIOGRAPHY

1. Evans, Thomas G., "A Program for the Solution of Geometric-Analogy Intelligence Test Questions," in Minsky (ed.), SEMANTIC INFORMATION PROCESSING, MIT Press, Cambridge, Mass., 1968. pp. 271-353.

2. Hewitt, Carl E., DESCRIPTION AND THEORETICAL ANALYSIS (USING SCHEMAS) OF PLANNER: A LANGUAGE FOR PROVING THEOREMS AND MANIPULATING MODELS IN A ROBOT, MIT, Department of Mathematics, Ph.D. dissertation.

3. Reinwald, Lewis, and Soland, Richard, "Conversion of Limited-Entry Decision Tables to Optimum Computer Programs I: Minimum Average Processing Time," Journal of the Association for Computing Machinery, July, 1966.

4. Sussman, Gerald Jay, and Winograd, Terry, "Micro-Planner Reference Manual," MIT, Artificial Intelligence Laboratory, Artificial Intelligence Memo No. 203, 1970.

5. Winograd, Terry, PROCEDURES AS A REPRESENTATION FOR DATA IN A COMPUTER PROGRAM FOR UNDERSTANDING NATURAL LANGUAGE, MIT, Project MAC , MAC-TR-84, Cambridge, Mass., 1971.

6. Winston, Patrick H., LEARNING STRUCTURAL DESCRIPTIONS FROM EXAMPLES, MIT, Project MAC, MAC-TR-76, Cambridge, Mass., 1970.